第2讲 | Exception和Error有什么区别?

2018-05-08 杨晓峰

第2讲 | Exception和Error有什么区别?

杨晓峰

00:00 / 11:14

世界上存在永远不会出错的程序吗?也许这只会出现在程序员的梦中。随着编程语言和软件的诞生,异常情况就如影随形地纠缠着我们,只有正确处理好意外情况,才能保证程序的可靠性。

Java语言在设计之初就提供了相对完善的异常处理机制,这也是Java得以大行其道的原因之一,因为这种机制大大降低了编写和维护可靠程序的门槛。如今,异常处理机制已经成为现代编程语言的标配。

今天我要问你的问题是,请对比Exception和Error,另外,运行时异常与一般异常有什么区别?

典型回答

Exception和Error都是继承了Throwable类,在Java中只有Throwable类型的实例才可以被抛出(throw)或者捕获(catch),它是异常处理机制的基本组成类型。

Exception和Error体现了Java平台设计者对不同异常情况的分类。Exception是程序正常运行中,可以预料的意外情况,可能并且应该被捕获,进行相应处理。

Error是指在正常情况下,不大可能出现的情况,绝大部分的Error都会导致程序(比如JVM自身)处于非正常的、不可恢复状态。既然是非正常情况,所以不便于也不需要捕获,常见的比如OutOfMemoryError之类,都是Error的子类。

Exception又分为可检查(checked)异常和不检查(unchecked)异常,可检查异常在源代码里必须显式地进行捕获处理,这是编译期检查的一部分。前面我介绍的不可查的Error,是Throwable不是Exception。

不检查异常就是所谓的运行时异常,类似 NullPointerException、ArrayIndexOutOfBoundsException之类,通常是可以编码避免的逻辑错误,具体根据需要来判断是否需要捕获,并不会在编译期强制要求。

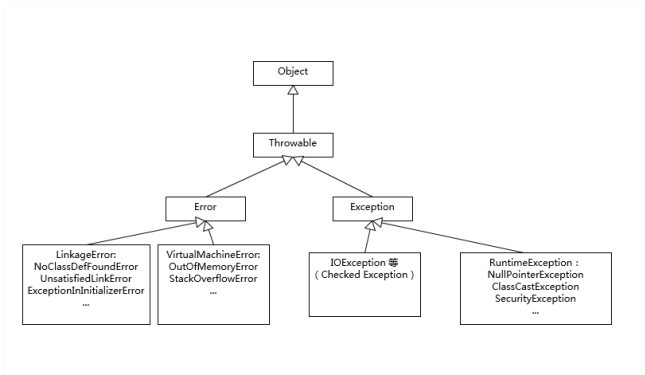
考点分析

分析Exception和Error的区别,是从概念角度考察了Java处理机制。总的来说,还处于理解的层面,面试者只要阐述清楚就好了。

我们在日常编程中,如何处理好异常是比较考验功底的,我觉得需要掌握两个方面。

第一,理解Throwable、Exception、Error的设计和分类。比如,掌握那些应用最为广泛的子类,以及如何自定义异常等。

很多面试官会进一步追问一些细节,比如,你了解哪些Error、Exception或者RuntimeException?我画了一个简单的类图,并列出来典型例子,可以给你作为参考,至少做到基本心里有数。



其中有些子类型，最好重点理解一下，比如NoClassDefFoundError和ClassNotFoundException有什么区别，这也是个经典的入门题目。

第二，理解Java语言中操作Throwable的元素和实践。掌握最基本的语法是必须的，如try-catch-finally块，throw、throws关键字等。与此同时，也要懂得如何处理典型场景。

异常处理代码比较繁琐，比如我们需要写很多千篇一律的捕获代码，或者在finally里面做一些资源回收工作。随着Java语言的发展，引入了一些更加便利的特性，比如try-with-resources和multiple catch，具体可以参考下面的代码段。在编译时期，会自动生成相应的处理逻辑，比如，自动按照约定俗成close那些扩展了AutoCloseable或者Closeable的对象。

```
try (BufferedReader br = new BufferedReader(…);
     BufferedWriter writer = new BufferedWriter(…)) {// Try-with-resources
// do something
catch ( IOException | XEception e) {// Multiple catch
   // Handle it
}
```

知识扩展

前面谈的大多是概念性的东西，下面我来谈些实践中的选择，我会结合一些代码用例进行分析。

先开看第一个吧，下面的代码反映了异常处理中哪些不当之处？

```
try {
  // 业务代码
  // …
  Thread.sleep(1000L);
} catch (Exception e) {
  // Ignore it
}
```

这段代码虽然很短，但是已经违反了异常处理的两个基本原则。

第一，尽量不要捕获类似Exception这样的通用异常，而是应该捕获特定异常，在这里是Thread.sleep()抛出的InterruptedException。

这是因为在日常的开发和合作中，我们读代码的机会往往超过写代码，软件工程是门协作的艺术，所以我们有义务让自己的代码能够直观地体现出尽量多的信息，而泛泛的Exception之类，恰恰隐藏了我们的目的。另外，我们也要保证程序不会捕获到我们不希望捕获的异常。比如，你可能更希望RuntimeException被扩散出来，而不是被捕获。

进一步讲，除非深思熟虑了，否则不要捕获Throwable或者Error，这样很难保证我们能够正确程序处理OutOfMemoryError。

第二，不要生吞（swallow）异常。这是异常处理中要特别注意的事情，因为很可能会导致非常难以诊断的诡异情况。

生吞异常，往往是基于假设这段代码可能不会发生，或者感觉忽略异常是无所谓的，但是千万不要在产品代码做这种假设！

如果我们不把异常抛出来，或者也没有输出到日志（Logger）之类，程序可能在后续代码以不可控的方式结束。没人能够轻易判断究竟是哪里抛出了异常，以及是什么原因产生了异常。

再来看看第二段代码

```
try {
   // 业务代码
   // …
} catch (IOException e) {
    e.printStackTrace();
}
```

这段代码作为一段实验代码，它是没有任何问题的，但是在产品代码中，通常都不允许这样处理。你先思考一下这是为什么呢？

我们先来看看printStackTrace()的文档，开头就是“Prints this throwable and its backtrace to the standard error stream”。问题就在这里，在稍微复杂一点的生产系统中，标准出错（STERR）不是个合适的输出选项，因为你很难判断出到底输出到哪里去了。

尤其是对于分布式系统，如果发生异常，但是无法找到堆栈轨迹（stacktrace），这纯属是为诊断设置障碍。所以，最好使用产品日志，详细地输出到日志系统里。

我们接下来看下面的代码段，体会一下Throw early, catch late原则。

```
public void readPreferences(String fileName){
     //...perform operations...
    InputStream in = new FileInputStream(fileName);
     //...read the preferences file...
}
```

如果fileName是null，那么程序就会抛出NullPointerException，但是由于没有第一时间暴露出问题，堆栈信息可能非常令人费解，往往需要相对复杂的定位。这个NPE只是作为例子，实际产品代码中，可能是各种情况，比如获取配置失败之类的。在发现问题的时候，第一时间抛出，能够更加清晰地反映问题。

我们可以修改一下，让问题“throw early”，对应的异常信息就非常直观了。

```
public void readPreferences(String filename) {
    Objects. requireNonNull(filename);
    //...perform other operations...
    InputStream in = new FileInputStream(filename);
     //...read the preferences file...
}
```

至于“catch late”，其实是我们经常苦恼的问题，捕获异常后，需要怎么处理呢？最差的处理方式，就是我前面提到的“生吞异常”，本质上其实是掩盖问题。如果实在不知道如何处理，可以选择保留原有异常的cause信息，直接再抛出或者构建新的异常抛出去。在更高层面，因为有了清晰的（业务）逻辑，往往会更清楚合适的处理方式是什么。

有的时候，我们会根据需要自定义异常，这个时候除了保证提供足够的信息，还有两点需要考虑：

- 是否需要定义成Checked Exception，因为这种类型设计的初衷更是为了从异常情况恢复，作为异常设计者，我们往往有充足信息进行分类。
- 在保证诊断信息足够的同时，也要考虑避免包含敏感信息，因为那样可能导致潜在的安全问题。如果我们看Java的标准类库，你可能注意到类似java.net.ConnectException，出错信息是类似“ Connection refused (Connection refused)”，而不包含具体的机器名、IP、端口等，一个重要考量就是信息安全。类似的情况在日志中也有，比如，用户数据一般是不可以输出到日志里面的。

业界有一种争论（甚至可以算是某种程度的共识），Java语言的Checked Exception也许是个设计错误，反对者列举了几点：

- Checked Exception的假设是我们捕获了异常，然后恢复程序。但是，其实我们大多数情况下，根本就不可能恢复。Checked Exception的使用，已经大大偏离了最初的设计目的。
- Checked Exception不兼容functional编程，如果你写过Lambda/Stream代码，相信深有体会。

很多开源项目，已经采纳了这种实践，比如Spring、Hibernate等，甚至反映在新的编程语言设计中，比如Scala等。 如果有兴趣，你可以参考：

http://literatejava.com/exceptions/checked-exceptions-javas-biggest-mistake/。

当然，很多人也觉得没有必要矫枉过正，因为确实有一些异常，比如和环境相关的IO、网络等，其实是存在可恢复性的，而且Java已经通过业界的海量实践，证明了其构建高质量软件的能力。我就不再进一步解读了，感兴趣的同学可以点击链接，观看Bruce Eckel在2018年全球软件开发大会QCon的分享Failing at Failing: How and Why We've Been Nonchalantly Moving Away From Exception Handling。

我们从性能角度来审视一下Java的异常处理机制，这里有两个可能会相对昂贵的地方：

- try-catch代码段会产生额外的性能开销，或者换个角度说，它往往会影响JVM对代码进行优化，所以建议仅捕获有必要的代码段，尽量不要一个大的try包住整段的代码；与此同时，利用异常控制代码流程，也不是一个好主意，远比我们通常意义上的条件语句（if/else、switch）要低效。
- Java每实例化一个Exception，都会对当时的栈进行快照，这是一个相对比较重的操作。如果发生的非常频繁，这个开销可就不能被忽略了。

所以，对于部分追求极致性能的底层类库，有种方式是尝试创建不进行栈快照的Exception。这本身也存在争议，因为这样做的假设在于，我创建异常时知道未来是否需要堆栈。问题是，实际上可能吗？小范围或许可能，但是在大规模项目中，这么做可能不是个理智的选择。如果需要堆栈，但又没有收集这些信息，在复杂情况下，尤其是类似微服务这种分布式系统，这会大大增加诊断的难度。

当我们的服务出现反应变慢、吞吐量下降的时候，检查发生最频繁的Exception也是一种思路。关于诊断后台变慢的问题，我会在后面的Java性能基础模块中系统探讨。

今天，我从一个常见的异常处理概念问题，简单总结了Java异常处理的机制。并结合代码，分析了一些普遍认可的最佳实践，以及业界最新的一些异常使用共识。最后，我分析了异常性能开销，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？可以思考一个问题，对于异常处理编程，不同的编程范式也会影响到异常处理策略，比如，现在非常火热的反应式编程（Reactive Stream），因为其本身是异步、基于事件机制的，所以出现异常情况，决不能简单抛出去；另外，由于代码堆栈不再是同步调用那种垂直的结构，这里的异常处理和日志需要更加小心，我们看到的往往是特定executor的堆栈，而不是业务方法调用关系。对于这种情况，你有什么好的办法吗？

请你在留言区分享一下你的解决方案，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。

公号-Java大后端

2018-05-08

在Java世界里，异常的出现让我们编写的程序运行起来更加的健壮，同时为程序在调试、运行期间发生的一些意外情况，提供了补救机会；即使遇到一些严重错误而无法弥补，异常也会非常忠实的记录所发生的这一切。以下是文章心得感悟：

1 不要推诿或延迟处理异常，就地解决最好，并且需要实实在在的进行处理，而不是只捕捉，不动作。

2 一个函数尽管抛出了多个异常，但是只有一个异常可被传播到调用端。最后被抛出的异常时唯一被调用端接收的异常，其他异常都会被吞没掩盖。如果调用端要知道造成失败的最初原因，程序之中就绝不能掩盖任何异常。

3 不要在finally代码块中处理返回值。

4 按照我们程序员的惯性认知：当遇到return语句的时候，执行函数会立刻返回。但是，在Java语言中，如果存在finally就会有例外。除了return语句，try代码块中的break或continue语句也可能使控制权进入finally代码块。

5 请勿在try代码块中调用return、break或continue语句。万一无法避免，一定要确保finally的存在不会改变函数的返回值。

6 函数返回值有两种类型：值类型与对象引用。对于对象引用，要特别小心，如果在finally代码块中对函数返回的对象成员属性进行了修改，即使不在finally块中显式调用return语句，这个修改也会作用于返回值上。

7 勿将异常用于控制流。

8 如无必要，勿用异常。

迷途知返

2018-05-17

我比较菜 在听到“NoClassDefFoundError 和 ClassNotFoundException 有什么区别，这也是个经典的入门题目。“ 这一段的时候 我以为会讲这两个的区别呢 我觉得这个区别详细讲讲 就是干货！文章总结性的语言比较多 并不具体

毛毛熊

2018-05-21

NoClassDefFoundError是一个错误(Error)，而ClassNOtFoundException是一个异常，在Java中对于错误和异常的处理是不同的，我们可以从异常中恢复程序但却不应该尝试从错误中恢复程序。
ClassNotFoundException的产生原因：

Java支持使用Class.forName方法来动态地加载类，任意一个类的类名如果被作为参数传递给这个方法都将导致该类被加载到JVM内存中，如果这个类在类路径中没有被找到，那么此时就会在运行时抛出ClassNotFoundException异常。

ClassNotFoundException的产生原因：

Java支持使用Class.forName方法来动态地加载类，任意一个类的类名如果被作为参数传递给这个方法都将导致该类被加载到JVM内存中，如果这个类在类路径中没有被找到，那么此时就会在运行时抛出ClassNotFoundException异常。
ClassNotFoundException的产生原因主要是：
Java支持使用反射方式在运行时动态加载类，例如使用Class.forName方法来动态地加载类时，可以将类名作为参数传递给上述方法从而将指定类加载到JVM内存中，如果这个类在类路径中没有被找到，那么此时就会在运行时抛出ClassNotFoundException异常。
解决该问题需要确保所需的类连同它依赖的包存在于类路径中，常见问题在于类名书写错误。
另外还有一个导致ClassNotFoundException的原因就是：当一个类已经某个类加载器加载到内存中了，此时另一个类加载器又尝试着动态地从同一个包中加载这个类。通过控制动态类加载过程，可以避免上述情况发生。

NoClassDefFoundError产生的原因在于：
如果JVM或者ClassLoader实例尝试加载（可以通过正常的方法调用，也可能是使用new来创建新的对象）类的时候却找不到类的定义。要查找的类在编译的时候是存在的，运行的时候却找不到了。这个时候就会导致NoClassDefFoundError.
造成该问题的原因可能是打包过程漏掉了部分类，或者jar包出现损坏或者篡改。解决这个问题的办法是查找那些在开发期间存在于类路径下但在运行期间却不在类路径下的类。

coder王

2018-05-08

留言中凸显高手。

钱宇祥

2018-05-08

1.异常：这种情况下的异常，可以通过完善任务重试机制，当执行异常时，保存当前任务信息加入重试队列。重试的策略根据业务需要决定，当达到重试上限依然无法成功，记录任务执行失败，同时发出告警。
2.日志：类比消息中间件，处在不同线程之间的同一任务，简单高效一点的做法可能是用traceId/requestId串联。有些日志系统本身支持MDC/NDC功能，可以串联相关联的日志。

作者回复

2018-05-08

很棒的总结

欧阳田

2018-05-08

1.Error:系统错误，虚拟机出错，我们处理不了，也不需要我们来处理。
2.Exception，可以捕获的异常，且作出处理。也就是要么捕获异常并作出处理，要么继续抛出异常。

3.RuntimeException，经常性出现的错误，可以
捕获，并作出处理，可以不捕获，也可以不用抛出。ArrayIndexOutOfBoundsException像这种异常可以不捕获，为什么呢？在一个程序里，使用很多数组，如果使用一次捕获一次，则很累。
4.继承某个异常时，重写方法时，要么不抛出异常，要么抛出一模一样的异常。
5.当一个try后跟了很多个catch时，必须先捕获小的异常再捕获大的异常。
6.假如一个异常发生了，控制台打印了许多行信息，是因为程序中进行多层方法调用造成的。关键是看类型和行号。
7.上传下载不能抛异常。上传下载一定要关流。
8.异常不是错误。异常控制代码流程不利于代码简单易读。
9.try catch finally执行流程，与 return，break，continue等混合使用注意代码执行顺序。不是不可以，而是越是厉害的人，代码越容易理解。

猿工匠

2018-05-08

每天早上学习与复习一下����

曹铮

2018-05-08

先说问题外的话，Java的checked exception总是被诟病，可我是从C#转到Java开发上来的，中间经历了go，体验过scala。我觉得Java这种机制并没有什么不好，不同的语言体验下来，错误与异常机制真是各有各的好处和槽点，而Java我觉得处在中间，不极端。当然老师提到lambda这确实是个问题...
至于响应式编程，我可以泛化为异步编程的概念嘛？一般各种异步编程框架都会对异常的传递和堆栈信息做处理吧？比如promise/future风格的。本质上大致就是把lambda中的异常捕获并封装，再进一步延续异步上下文，或者转同步处理时拿到原始的错误和堆栈信息

作者回复

2018-05-08

是的，非常棒的总结，归根结底我们需要一堆人合作构建各种规模的程序，Java异常处理有槽点，但实践证明了其能力；
类似第二点，我个人也觉得可以泛化为异步编程的概念，比如Future Stage之类使用ExecutionException的思路

Alphabet

2018-05-10

老师可以在文章末尾推荐一些基础和进阶的Java学习书籍或是资料吗？最好是使用较新版本jdk的

涟漪

2018-05-09

非常感谢作者以及评论中的高手们！我很喜欢作者能够精选评论。

飞云

2018-05-08

能不能讲下怎么捕捉整个项目的全局异常，说实话前两篇额干货都不多，希望点更实在的干货

作者回复

2018-05-08

谢谢建议，极客课程设计是尽量偏向通用场景，我们掉坑里，往往都不是在高大上的地方；全局异常Spring MVC的方式就很实用；对与干货，你是希望特定场景，特定问题吗？说说你的想法

小绵羊拉拉

2018-05-08

看完文章简单认识一些浅层的意思 但是我关注的 比如try catch源码实现 涉及 以及 文章中提到 try catch 产生 堆栈快照 影响jvm性能等 一笔带过 觉得不太过瘾。只是对于阿里的面试 读懂这篇文章还是不够。还希望作者从面试官的角度由浅入深的剖析异常处理 最后还是 谢谢分享

作者回复

2018-05-09

谢谢反馈，如果不做jvm或非常底层开发，个人没有看到这些细节的实际意义，如果非要问可以鄙视他：-）
创建Throwable因为要调用native方法fillInStacktrace；至于try catch finally，jvms第三章有细节，也可以自己写一段程序，用javap反编译看看 goto、异常表等等

Jerry银银

2018-05-08

由于反应式编程是异步的，基于事件的，所以异常肯定不能直接抛出，如果直接抛出，随便一个异常都会引起程序崩溃，直接影响到对后续事件处理。个人觉得一种处理方式是：当某个事件发生异常时，为了不影响对后续事件的处理，可以对当前发生异常的事件进行拦截处理，然后将异常信息发送出去。

至于发生异常时，堆栈信息只是关于特定executor框架中的，不知道是否可以将之前事件的"上下文"带到executor，再传递给观察者？
（对反应式编程不太了解，尝试作答^_^）

James

2018-05-08

个人觉得checked exception / unchecked exception 分别翻译为 检查型异常/非检查型异常 更加好理解。
可检查异常容易被理解为可以不检查。

作者回复

2018-05-08

有道理，谢谢指出

三军

2018-05-10

Java语言规范将派生于Error类或RuntimeException类的所有异常称为未检查（unchecked）异常，所有其他的异常成为已检查（checked）异常。

编译器将检查是否为所有的已检查异常提供了异常处理器。

这是经典，要好好理解。

平时我们使用throws往外抛错，或者try-catch这类异常处理器不就是处理已检查异常吗��

未检查异常就是潜在的，编译器无需提供异常处理器进行处理。

米斯特.杜

2018-05-08

Java 每实例化一个 Exception，都会对当时的栈进行快照，这是一个相对比较重的操作。如果发生的非常频繁，这个开销可就不能被忽略了。

提问：为什么要生成快照，什么时间销毁呢？

whhbbq

2018-05-09

关于检查型异常，因为要强制捕获或者在函数签名声明，导致要写好多的代码。现在都改为用运行时异常，根据业务定义好编码和消息，然后通过全局的异常处理器处理。一些特殊的需要携带更多信息的异常，会自定义异常类，当然它也是运行时异常。

张世杰

2018-05-08

老师总结的类图，对理解Throwable,Exception,Error非常的直观！但再说掌握的两个方面，第二方面时候，仅仅提到懂得如何处理典型场景！如果能详细描述一下什么样的典型场景，会对深入理解，使用Exception,Error非常有帮助！

五年

2018-05-24

老师讲的很好 ��

不过理论讲过之后很容易忘 老师可以开一个github的代码库，将课程的最佳实践还有反例放进去吗

作者回复

2018-05-25

有打算，最近出差，黑白颠倒，回去找机会弄下

DavidWhom佳传

2018-05-14

提出面试问题，却没有较好的回答，很难受(;_;)

Ccook

2018-05-10

大佬能介绍下，线程间调用导致异常信息丢失的问题吗

作者回复

2018-05-10

Thtead 一个UnCaughtExceptionHandler

暴走的��

2018-05-09

业务规则检验是抛异常好还是if return好

作者回复

2018-05-09

我在文章里提到了,不建议以异常控制业务流程

---

fangxuan

2018-05-08

有种浅尝辄止的感觉，希望老师能再深入一点。另外对于程序中到底是该抛出异常还是默默的处理掉，把我不好，希望老师能给点这方面的最佳实践。函数式编程中遇到checkedexception怎么处理能详细指点一下吗?

13683815260

2018-05-08

个人想的有三步:
1，完善的异常记录。包括调用的上下文信息，如果在同一个进程内考虑ThreadLocal传递参数。如果分布式，把核心的参数封装传递。
2，在基础1之上构建traceid之类的调用链跟踪。
3，基于回调机制，发生异常时以事件的方式通知调用方。

另 对学习的的结果做个小总结。首先二者继承体系的异同。设计里面也不同，error一般表示不可自主从异常中恢复，Exception意味着可能可以恢复。其中Exception分为两类检查异常，非检查异常。
最佳实践，try中的代码块不宜过长，捕获时不宜大而全，finally里只释放资源不要有业务逻辑，尤其是修改返回值。用新语法可增强代码的可读性和简洁性。
业务异常可继承runntimeexception，封装applicationexception.
finally中的代码始终是执行的，用途为清理资源。
对于线程池注意runntimeexception导致的线程逃逸现象。

阿修罗哇

2018-05-08

对于日志里面我们看到的往往是特定 executor 的堆栈，而不是业务方法调用关系这种情况，我在公司推行的是自定义异常，自定义的异常有一个错误码，这个错误码需要细到某个业务的某个方法的某种错，这样排查问题会很方便，但是写的时候就比较麻烦，文档也比较多

作者回复

2018-05-08

嗯，有些类似trace id的思路，构建树形堆栈也有帮助

风动静泉

2018-05-08

"Exception 又分为可检查（checked）异常和不检查（unchecked）异常"这句话本身没问题，但是不够全面吧。查了下<<JAVA核心技术 卷Ⅰ>> 第9版，pp.474 "JAVA语言规范将派生于Error类或RuntimeException类的所有异常称为未检查(unchecked)异常，所有其他的异常称为已检查(checked)异常。"

作者回复

2018-05-08

没错，看描述的角度和范围，不然让人晕了

拉灯灯

2018-05-16

error指的是不可预料的错误，可能会导致程序宕机；而exception指的是在程序运行中可以预见的异常，而异常分为检查异常与一般异常，检查异常需要在程序中显示捕获并处理，一般异常可以通过程序编码来进行处理，比如数组越界，空指针等；异常处理的两大基本原则：不要捕获泛泛的异常信息，比如直接捕获Exception，这样会在增加代码阅读难度；不要生吞异常；打印异常信息是一个比较重的操作，会导致程序变慢；try catch最好是包括需要检验异常的代码，不要包含过长代码，这样会降低JVM的优化效率；这是学习本节课的部分总结

---

zero

2018-05-16

如果业务中有一个段业务逻辑抛出异常，不能影响后面的业务逻辑的处理，如果不用try catch 吃掉异常，还有什么好的办法处理呢?

---

happyhacking

2018-05-15

提几点我觉得可以改进的地方，
简单的内容可以篇幅少一些
标题和结构感觉不够清晰
结合实践可以更多一些

我的感受是，一篇读完，读之前模棱两可的，读之后还是不清楚，如果不回头再看一遍的话都不记得有什么内容...

对了，可以贴一些推荐的好的文章和资料。总之让读者看到用心呀。
否则和网上搜到的资料水准不就差不多了~
不过本来就基础的东西，要体现出水平还不能说太深又要实用还是挺难的。加油

---

sonnyfu

2018-05-12

捕获了throw或error，为啥就难以保证我们能够正确程序处理 OutOfMemoryError?

作者回复

2018-05-12

请问抓住OOM，写什么代码处理，如果内存已经over commit，怎么保证后续逻辑可靠性，不是做不到，不大容易

不吃老鼠的猫

2018-05-10

看了整篇文章和留言，大家都提到了不能用异常控制流程，这个我也懂，可是在项目中比如一个serivce方法，会对请求参数做检验，如果请求参数bean有5个属性需要检验，检验不成功，怎么处理？我目前项目中大都是如果检验不成功，就throw一个RuntimeException，如果不用这种抛异常的方式，用其他什么方式让上层调用放知道呢？如果用返回值，是不是要定义好多返回码?

作者回复

2018-05-10

这种处理是对的，这不是通常意义的业务逻辑

石头狮子

2018-05-09

1，无论各种异常均可以看成线程无法继续执行的信号(引发线程中断)。是否恢复执行由用户决定。信号的重要程度构成了 Throwable 的继承层级。
2，既然是线程无法继续执行就需要打印线程的执行状况。以便分析。
3，既然是线程中断，就可以跨调用 catch 与throw 而不用像 c 等过程语言要判断每个函数的返回。方便统一 catch 处理。
不足之处望指出。

雷霹雳的爸爸

2018-05-09

executor 出来的异常和外层逻辑的的关联信息可以考虑实例化线程池时候自定义threadfactory保留一部分，比如线程名称前缀在日志里就蛮有用的，而且扩展这个factry还有一个有用的地方在于可以处理那些未补货的异常，比如调用的底层代码的运行时异常，老师讲到的那个被很多框架尊崇的一切都该是运行时异常的哲学往往会让人多线程时候在这个点崴脚，不过呢，这里也有一大招，就是可以"深思熟虑"一下，考虑使用个反模式，Runnable时候try一个大的throwable，然后catch里面记个log来避免这种意外中异常被吞的情况，坏处是老师课程里提到的执行效率的代价，而且直接和工作代码耦合，当然也可以中间加一个油漆工模式隔离下这个try块，但就没了这个反模式的唯一好处：仅看run方法上下文就知道这东西是不是已经处理了，之所以这里可以考虑违反通常对异常的最佳实践，主要是要看你处理的问题规模和粒度，这会影响最终你测量出来的性能的差异是不是足够明显，和让你团队中绝大多数人快速理解排查问题哪一个你更在意，本质上看，如果是团队开发，应该有一致的风格，我倾向于前者，因为这个问题完全可以结合代码审查和静态代码检查工具来做，形成一个统一的团队的代码风格，如果自己干私活...一般可能就不用java了哈

梁中华

2018-05-08

关于捕获全局异常，可以考虑使用AOP技术在接口入口层统一捕获，特别是使用类似dubbo这样的非springmvc架构的系统非常有用

作者回复

2018-05-08

嗯，算是切面编程典型场景

dingsai88

2018-05-08

物超所值啊 买的很对

作者回复

2018-05-08

感谢认可，很高兴对大家有帮助

BUGS

2018-05-08

Spring cloud sleuth不知道是否可以解决调用跟踪问题（包括异常？）

作者回复

2018-05-08

有帮助

Hesher

2018-05-08

首先说说异常，虽然所有人都不建议一个大try-catch包住整个方法捕获exception，但我还是不放心啊，怎么破？我的做法是对小的代码段try-catch捕获指定的异常，然后在最外面套一个大的try-catch防止遗漏的情况，不知道这样做算不算一个折衷的办法，希望晓峰老师和其他同学多多指教。

响应式编程接触比较少，没什么经验，笨办法还是有。异常还是那些异常，如果把握不好捕获的位置和方法，不妨前期多加一些日志，把关键参数输出出来，从错误中反向总结异常处理方式。

作者回复

2018-05-08

都抓起来是担心RuntimeException吗？即使抓起来，业务逻辑怎么走下去呢？
对于日志，实用主义也不错啊

YANGFEI

2018-05-08

Checked Exception 不兼容 functional 编程，如果你写过 Lambda/Stream 代码，相信深有体会。这段话可以详细剖析下吗？

作者回复

2018-05-08

比如，写段lambda的程序试试，调用一个抛出检查异常的方法，现在语言层面并没有流畅处理的机制，往往需要自定义functional interface

Dean

2018-07-17

在使用RxJava或者有重试机制的框架时，许多调用是通过线程池方式运行，那么这时的异常只能由当前执行线程捕获，可以通过写日志的方式记录，并通过traceid与主线程关联，当然traceid的传递也是需要格外注意的。

Patrick

2018-07-16

老师可不可以做个Github repo 把样例代码可以让大家动动手，谢谢老师的课

feifei

2018-06-29

对于此场景处理分为几个部分
1，发生异常时，将异常信息收集到统一的日志中，不是直接的处理，然后在日志中心进行日志的查看
2，对于任务根据业务定义重试机制！
3，业务线程要独立与reactor线程

这是我的观点，欢迎指正，谢谢

ZK

2018-06-28

我就想问，很多程序包括开源的都是比如参数错误，参数非法 类型错误，不进行判断，直接throw出来异常在外层进行捕获异常后整体返回出去，肯定对性能有影响吧？那么为什么很多还这样做呢？如何取舍？或者优化？

张小小的席大da

2018-06-27

感谢老师的分享 感谢下方评论的小伙伴的全面解析

荣

2018-06-26

checked exception / unchecked exception是相对编译器（javac）而言，可检查和不可检查的到.

带着猪散步

2018-06-25

可是我要执行一连串流程，每个中断就随时结束返给前端，并写mq接着重试，这时用异常抛出去更好控制，况且他执行到这有问题了，确实也算异常，这时可以这么搞吧

张焕旭

2018-06-22

try catch块真的会影响性能吗？我记得java编程思想里面提到并不会影响性能，请尽情使用

---

robbin��

2018-06-12

我们在实际开发中因为不能准确保证程序的哪里会有异常情况，一般都会将代码段包起来，同时使用throwable捕获，有时很难做到每段都仔细判断异常。

关于异步的部分我想是否可以像golang的模式将上下文对象传递

洺叶

2018-06-02

Printstacktrace 在稍微复杂一点的生产系统中，很难判断出到底输出到哪里去了，老师能否举例说明下会输出到什么地方？

作者回复

2018-06-03

默认是标准输出，具体看应用重定向到哪里

韩峰

2018-05-30

老师，关于异常，我有两点疑惑，第一点一个程序一级级的调用，是应该在在最上级捕获吗，第二点，目前我公司已有的项目自定义大量的业务异常继承自runtimException,比如库存不足异常，这样会不会额外增加开销，这是不是一种不可理的方式

作者回复

2018-06-01

这个建议要从业务逻辑角度看哪里负责处理，比如你提到的基础不足，我相信是业务逻辑的一部分，如何处理其实从业务设计角度是有答案的，例如，应用肯定是要给用户一个合理的反馈，而不是简单打个堆栈或不响应。
开销不用过于担心，毕竟也不是netty这种特别性能敏感的框架，还是优先考虑业务需要

Jerome

2018-05-28

如果项目自定义异常应该继承exception还是runtimeexception，我现在写的代码两种都有， 这两种形式有什么区别 比如错误显示，记录日志。什么场景下用呢？

徐金铎

2018-05-26

从架构，或者不同模块的角度，推荐大家注意一个点，一般类似mybatis这类的框架，都会有关于exception的converter，比如mybatis会先封装mysqlexception，然后是自己的，再封装向上是spring的exception。类似的思想在微服务调用链路也有体现，上游服务对下游服务的error解析convert，是可以加强代码健壮性的。

密码123456

2018-05-25

1.异常的父类。throwable.
2.异常的分类。error错误。exception异常。
3。对异常分类的使用。error是jvm环境运行错误，不可进行捕获。包括，throwable。exception是程序上的错误，需要在错误时进行捕获，恢复正常运行的形态。对异常捕获，最好进行异常类型最匹配的形式，这样具有日志堆栈信息便于查询，排错。
4。异常的使用是比较消耗性能。消耗性能的方式有try代码快，与生成exception的堆栈快照。
5。注意信息。在生成exception错误信息时，不能使用exception自带方法进行输出。这种方式，不清楚会输出到什么地方，不好排查。
6。异常实践。异常分2种进行处理，一种业务异常，一种程序异常。业务异常直接抛出，程序异常，先处理一次，如果处理不了在进行抛出。

---

aoe

2018-05-24

请问老师，当catch 住异常时
catch (IOException e) {
e.printStackTrace();
}
1、正确的打印异常的方式是什么？
例如使用slf4j：
log.error("发现异常了 {}", e.getMessage());
2、日志的级别应该是warn、error的哪一个更合适？
谢谢！

---

aoe

2018-05-24

最大的收获是明白了为什么不建议用异常控制正常业务流程，因为这种条件判断方式是低效的。具体原因：Java 每实例化一个 Exception，都会对当时的栈进行快照，这是一个相对比较重的操作。如果发生的非常频繁，这个开销可就不能被忽略了。

---

azhansy

2018-05-21

我们读代码的机会往往超过写代码，
软件工程是门协作的艺术，
优秀是一种习惯。

周红阳

2018-05-20

使用Exception的性能问题是两个方面：1. 创建exception,尤其是fillInStackTrace开销巨大；2. 部分编译器优化策越不能使用. 第一个问题解决方法有2个：1, cache exception,(或者Exception设计为单例)，他的问题是stackTrace打出来是错误的. 2, 禁止掉fillInStackTrace操作,问题是stackTrace完全没有了,不方便定位问题. 可以在系统上线稳定后,使用者这个办法. 第二个问题可以忽略，可读性好,维护性好的代码比性能重要很多.□

try catch NullPointerException和使用if(ref!=null)，使用exception只有exception实际发生的时候才会有问题一的性能问题；但是这个if判断每次都要执行. 异常情况毕竟是极少数. 平衡性能问题也要考虑到. 而且并不是每个代码都有编译器优化

我认为：exception写的代码更简洁,逻辑更清晰,我喜欢用.

马婷婷

2018-05-19

NoClassDefFoundError 和 ClassNotFoundException 有什么区别

前者是找不到类的定义，类文件还在；后者是找不到class文件；前者可能是在类初始化的时候（比如初始化某个变量报错）发生异常；后者可能是找不到引用的类文件，可能是某个包没有导入.前者是运行时异常，后者是可检查异常

日光倾城

2018-05-19

Exception与Error都继承自Throwable，单纯从字面理解，前者是异常，后者是错误。Exception是在编码过程中可以预见的异常情况，比如数组越界，文件不存在，网络不通等。Error是正常情况下不会发生，一旦发生jvm也无法进行恢复的情况，比如OutOfMemoryError，StackOverflowError，这些错误情况一旦出现，jvm也无能为力了，只能停止整个进程。
运行时异常也就是非受检异常，比如说NPE，数组越界，这种异常其实是我们可以通过一些判断避免的，比如非空判断，数组长度判断，如果我们没做判断抛出了这类异常，说明我们的程序不严谨。一般异常就是我们在编码阶段无法预知但是也无法避免的异常情况，比如网络不通之类，所以我们可以提前进行预防，发生这种情况我们如何处理？并尝试从这类异常中恢复，至少能给出良好的提示

日光倾城

2018-05-19

Exception与Error都继承自Throwable，单纯从字面理解，前者是异常，后者是错误。Exception是在编码过程中可以预见的异常情况，比如数组越界，文件不存在，网络不通等。Error是正

常情况下不会发生，一旦发生jvm也无法进行恢复的情况，比如OutOfMemoryError，StackOverflowError，这些错误情况一旦出现，jvm也无能为力了，只能停止整个进程。
运行时异常也就是非受检异常，比如说NPE，数组越界，这种异常其实是我们可以通过一些判断避免的，比如非空判断，数组长度判断，如果我们没做判断抛出了这类异常，说明我们的程序不严谨。一般异常就是我们在编码阶段无法预知但是也无法避免的异常情况，比如网络不通之类，所以我们可以提前进行预防，发生这种情况我们如何处理？并尝试从这类异常中恢复，至少能给出良好的提示

老男孩

2018-05-18

每课都提出课后问题，这种反馈式的学习方法有助于加深理解。而且课后的留言还是主题的一个很好补充。����

New Yorker

2018-05-16

两点疑问：
1，checked exception 和 unchecked exception 是 Exception 的子类吗？如果不是，如何自定义一个unchecked exception？
2，如何控制一个Exception 在实例化时不进行栈快照？

作者回复

2018-05-18

1，狭义上是，Error也是unchecked类
2，比如cache一个，或者自己实现个，去掉对应逻辑

戴

2018-05-16

留言区的一些留言，语言表达能力好差，很难看懂。

戴

2018-05-16

多看看评论，感觉评论比原文有价值。评论里更像一个专题交流会，从中可以看到大家日常工作实践中面临的痛点

LBF

2018-05-16

小白留言：
捕获不了异常的捕获不是好捕获，
什么异常都捕获的捕获不是好捕获，
解决不了异常的捕获一定是坏捕获，
捕获不是俄罗斯转盘，小概率事件会致命！

先天专治不服

2018-05-15

同样有一个问题，在spring mvc中，就提供了controllerAdvice注解，在业主流程中抛出相应的异常，在controllerAdvice中通过异常类型，完成相应的message输出。这个本质上也是通过异常完成对流程控制。而且能让业务流程很干净整洁。但是我一直很纠结，按文中说法，这是比较低效的流程控制，请问要如何取舍？

作者回复

2018-05-15

区分它是用来作为业务流转，还是利用切面编程做异常处理？
再说所谓实践，本就是仁者见仁的事

落叶飞逝的恋

2018-05-14

只看到理论介绍，和一些实验代码，没看到生产代码实践示范？还比如，如何合理自定义异常的示范？

末日没有进行曲

2018-05-13

问一下老师如何能更好地消化这些知识呢？感觉看过一遍懂，但是自己说却不能完整表述出来。多看？

---

ls

2018-05-13

老师您好，我是做Android 客户端的,对于网络请求回来的bean 解释及后续处理，我们的做法也是把一整段给try-catch，即使有异常没有被处理，也不会导致客户端崩溃 。像客户端这种频率不是很大的，对性能影响会怎么样？

另外也很希望能讲解下自定义异常的实践，或者如何去设计一个合适的异常处理机制。

作者回复

2018-05-14

“过早优化是万恶之源”，当然是保证应用可靠性更重要

---

davidimu

2018-05-12

RxJava中有一些异常处理的方法 比如doOnError switchOnError 可以选择把异常包起来再抛出去来保留上下文

作者回复

2018-05-14

Rxjava我了解有限，如果我没理解错，那是reactive编程注册异常listener的方式，jdk最新版本websocket api也是使用类似机制，虽然细节有区别

云帆

2018-05-12

有些吃力呀

清风

2018-05-12

我有一个问题想请教一下，目前公司里的异常处理方式是直接用try catch 把整个逻辑块都包住，不管是controller层还是service层，这种代码到处可见，虽然觉得这样不太好，但确没有想到更好的解决方案，因为这样确实比较容易排查问题。而且目前使用的是微服务式开发，每个功能可能都由不同人开发，服务调用采用不信任原则也是有道理的，所以想请问一下，目前有关于在这样情况下比较好的异常处理方案吗？补充一下，公司用的是spring boot框架。

作者回复

2018-05-14

保证可靠性是最重要的；这种情况使用了spring exception handler吗？

自在

2018-05-11

我记得栈帧里面异常块与代码块是分离的，如果不出现异常好像是不影响性能的吧...

作者回复

2018-05-11

也会影响优化

淡云天

2018-05-10

在忽略try-catch和流程控制时的性能消耗下，前者的控制能力更强一些吧？一些业务级异常都是交给顶层的ExceptionHandler处理的，写一个整体处理类即可。而用流程控制的话，细节处理上就会相当麻烦。而且像servlet等一些类似server的程序，都会在外面包一层try-catch以保证程序在非致命Error下可以正常运行的吧？还望作者解惑

作者回复

2018-05-10

符合catch late原则，业务代码更清楚怎么处理，这个看什么角度，不完全通常意义的流程控制；
"非致命"Error的出现，恰恰说明了异常机制在实践中并没有达到设计目的，所以才出现了争论，这么catch前提是清楚那些Error是应用能处理的

Gerald

2018-05-10

"尤其是对于分布式系统，如果发生异常，但是无法找到堆栈轨迹（stacktrace），这纯属是为诊断设置障碍。所以，最好使用产品日志，详细地输出到日志系统里。"什么是产品日志呢？

作者回复

2018-05-10

应用里的logger

Gerald

2018-05-10

e.printStackTrace();在产品代码里使用有问题，这个不是能理解，IOException 是checked 异常，在catch块里应该怎么处理呢？

小马

2018-05-10

学习总结：1.理解Throwable,Error,Exception的关系。
2.尽量不要捕获Exception通用异常,这样会使程序保留隐藏的异常，有很大的隐患。
3.不能生吞异常，就是将一大段代码放在try中，不知道会发生那些异常，就将其全部包起来，问题一保留隐藏异常对程序有隐患问题二：异常捕获处理对jvm的开销很大。更不能在异常处理实现业务逻辑。
学会了规范写异常处理部分的代码，以及这么规范的好处。

YI

2018-05-10

复习了一下Java异常处理知识，有很多启发性的内容，受益匪浅！最后的思考题我也想了下，是否可以构建一个独立的异常分析模块，业务端在提交异步任务时，将其重要的状态信息及traceid发送到异常分析模块，同时异步任务也与traceid捆绑，发生异常时，异步任务只需把捆绑的traceid及异常信息发送给异常分析模块，由异常分析模块来处理业务端与异常的配对。

coulson

2018-05-09

老师，你写的这些我大部分都能看懂，也能理解一部分，可是让我用文字形式写出来，我脑子就一踏糊涂，不知道从哪下手，该怎么办？

作者回复

2018-05-09

心动不如行动，个人建议，试着问自己一个特定的问题，然后针对性的总结，别怕不够完美。
编程也是一样，不是什么时候都完全想清楚再做的

无意中找到的

2018-05-09

同事老是在catch中，直接用e.print打印堆栈，然后再用log打印一个e.getMessage() 还说在log里面没必要把堆栈打出来，太浪费空间时间了。。。感觉这样明显有问题，可我就是说不过他-_-||

邓志国

2018-05-09

checked exception至少有一个用，通过编译时提醒你如果出错应该释放资源。否则如果是unchecked，说不定忘记了

Mine

2018-05-09

最近学了一点kotlin，在kotlin里面就是没有checkedException。这里想请教一下老师，在这种一切都是runtime的情况下应该以何种思路处理异常？因为不知道调用的函数会不会抛异常，不知道会抛哪种异常，而都用try catch包起来也不现实，最近一直在困惑这个点.....

作者回复

2018-05-09

我对Kotlin了解有限，初略的理解是：
二者不完全对比，因为kotlin catch可以返回值，try-catch就变成了表达式，更优雅一些；
对于函数会不会抛异常，IDE不会提供提示吗

渊

2018-05-09

如何知道我的try bllck会throw什么exception?

渊

2018-05-09

我怎么知道我try block应该catch什么exception?

作者回复

2018-05-09

try住的代码，抛出什么是明确声明了的，这种情况一般IDE更擅长

wang_bo

2018-05-09

error开发者能不能去捕获和处理？

作者回复

2018-05-09

绝大部分情况是不，除非确定可以从Error恢复，或者没有别的选择

大胖

2018-05-09

error:jvm或者说是系统级别问题，捕获会导致问题无法定位，而且错误捕获应用程序也没什么作为，不如暴露出来定位错误！
exception:RuntimeException以外的子类,属于应用程序可以处理的也在编程过程中可以遇判的，应该抛出异常进行处理！这样理解对否？

沈老师

2018-05-09

能否选择您项目中一段设计比较完善的异常处理结合业务分析下，这样比较有具体场景的深入了解好的代码是怎么一个思路。谢谢！

板砖

2018-05-09

不要用一个大的try-catch包含整个代码这个怎么理解呢，因为springmvc来说，我现在的做法是在axtion里一个方法写一个业务，然后就写一个try-catch，如果不用一个大的该怎么写，还是说老师您的说法就是一个业务一个try-catch？？？

VincentWei

2018-05-08

链路追踪

愣子

2018-05-08

反应式编程里出现异常是否可以搞一个类似dead queue的异常队列，把异常放到这个队列里供消费者去消费？

我奋斗去了

2018-05-08

Reactor 目前在Java中还没有使用过 ，感觉类似nodeJs的回调？期待正解

Rebby

2018-05-08

try 包住整段的代码；远比我们通常意义上的条件语句（if/else、switch）要低效。
请问 这段怎么理解呢？

作者回复

2018-05-08

有的代码被设计为，当发生某种异常时，走某条路径，这种条件判断方式是低效的

Hidden

2018-05-08

分析的太好了，学习完，基础就牢固很多了

作者回复

2018-05-08

很高兴能有帮助，程序开发不是只有那些高大上的东西的，以前有人找我帮助诊断应用上线闪退问题，我发现大部分就是几个NPE、IllegalState之类runtime exception低级错误导致的！

刹那间的永恒

2018-05-08

由于是半路出家，所以对于基础这块儿很不牢靠，尤其是之前对于异常处理很不重视，看了这篇文章收获了很多：
1.catch具体的异常，不是exception；
2.慎用try catch，只包裹可能会抛出异常的代码；
3.慎用return break continue；
4.慎用e.printstacktrace()，都记录到日志(这个还不是很理解)；
5.能处理的异常尽早捕获处理，不能的就抛给上层。

作者回复

2018-05-08

换个角度，如果出了错，我们是希望看看log就能定位问题，还是觉得需要熬夜改改代码多print几行？
前人（自己）挖坑，含泪也得爬出来

Heart to bodhi

2018-05-08

对于反应式编程 应该用日志记录好每个 executor信息 和输入信息

李志博

2018-05-08

Reactor我记得有onError和doError开头的一些方法

作者回复

2018-05-08

对头，非常不同的编程模式

不会游泳的鱼

2018-05-08

exception做设计的话有什么好的设计方案么？自动测试中捕获异常是必备的

峰

2018-05-08

扩展任务类的实现，把当前的堆栈信息保留进去后提交，这样在任务中捕获异常后就可以做相关处理。