Java异常处理的陋习展播

2013/07/01 • 开发 • 7 评论 • java, 异常

注：本文来自志军（@_Zhijun ）在微博推荐的一篇转载于2007年09月27的旧文，他说“没法找到原作者”。的确，我也花了半个多小时在找原作者。先是找到了一个标注“2005年6月18日转字Java研究组织”的文章（“Java研究组织”的域名已过期），后来找到是大连理工大学碧海青天BBS上一个发于的2003年5月中旬的合集帖子，其中提到一条来自 CSDN 的链接，可惜该链接已挂，否则应该能找到作者的。

你觉得自己是一个Java专家吗？是否肯定自己已经全面掌握了Java的异常处理机制？在下面这段代码中，你能够迅速找出异常处理的六个问题吗？

```
1  OutputStreamWriter out = ...
2  java.sql.Connection conn = ...
3  try {
4  // (5)
5     Statement stat = conn.createStatement();
6     ResultSet rs = stat.executeQuery(
7       "select uid, name from user");
8     while (rs.next())
9     {
10      out.println("ID: " + rs.getString("uid")
11 // (6)
12        ", 姓名: " + rs.getString("name"));
13    }
14    conn.close();
15 // (3)
16    out.close();
```

```
17    }
18    catch(Exception ex)
    // (2)
      {
        ex.printStackTrace();
    //(1),(4)
      }
```

作为一个Java程序员，你至少应该能够找出两个问题。但是，如果你不能找出全部六个问题，请继续阅读本文。

本文讨论的不是Java异常处理的一般性原则，因为这些原则已经被大多数人熟知。我们要做的是分析各种可称为“反例”（anti-pattern）的违背优秀编码规范的常见坏习惯，帮助读者熟悉这些典型的反面例子，从而能够在实际工作中敏锐地察觉和避免这些问题。

反例之一：丢弃异常

代码：15行-18行。

这段代码捕获了异常却不作任何处理，可以算得上Java编程中的杀手。从问题出现的频繁程度和祸害程度来看，它也许可以和C/C++程序的一个恶名远播的问题相提并论??不检查缓冲区是否已满。如果你看到了这种丢弃（而不是抛出）异常的情况，可以百分之九十九地肯定代码存在问题（在极少数情况下，这段代码有存在的理由，但最好加上完整的注释，以免引起别人误解）。

这段代码的错误在于，异常（几乎）总是意味着某些事情不对劲了，或者说至少发生了某些不寻常的事情，我们不应该对程序发出的求救信号保持沉默和无动于衷。调用一下printStackTrace算不上“处理异常”。不错，调用printStackTrace对调试程序有帮助，但程序调试阶段结束之后，printStackTrace就不应再在异常处理模块中担负主要责任了。

丢弃异常的情形非常普遍。打开JDK的ThreadDeath类的文档，可以看到下面这段说明：“特别地，虽然出现ThreadDeath是一种‘正常的情形’，但ThreadDeath类是Error而不是Exception的子类，因为许多应用会捕获所有的Exception然后丢弃它不再理睬。”这段话的意思是，虽然ThreadDeath代表的是一种普通的问题，但鉴于许多应用会试图捕获所有异常然后不予以适当的处理，所以JDK把ThreadDeath定义成了Error的子类，因为Error类代表的是一般的应用不应该去捕获的严重问题。可见，丢弃异常这一坏习惯是如此常见，它甚至已经影响到了Java本身的设计。

那么，应该怎样改正呢？主要有四个选择：

1、处理异常。针对该异常采取一些行动，例如修正问题、提醒某个人或进行其他一些处理，要根据具体的情形确定应该采取的动作。再次说明，调用printStackTrace算不上已经“处理好了异常”。

2、重新抛出异常。处理异常的代码在分析异常之后，认为自己不能处理它，重新抛出异常也不失为一种选择。

3、把该异常转换成另一种异常。大多数情况下，这是指把一个低级的异常转换成应用级的异常（其含义更容易被用户了解的异常）。

4、不要捕获异常。

结论一：既然捕获了异常，就要对它进行适当的处理。不要捕获异常之后又把它丢弃，不予理睬。

反例之二：不指定具体的异常

代码：15行。

许多时候人们会被这样一种“美妙的”想法吸引：用一个catch语句捕获所有的异常。最常见的情形就

是使用catch(Exception ex)语句。但实际上，在绝大多数情况下，这种做法不值得提倡。为什么呢？

要理解其原因，我们必须回顾一下catch语句的用途。catch语句表示我们预期会出现某种异常，而且希望能够处理该异常。异常类的作用就是告诉Java编译器我们想要处理的是哪一种异常。由于绝大多数异常都直接或间接从java.lang.Exception派生，catch(Exception ex)就相当于说我们想要处理几乎所有的异常。

再来看看前面的代码例子。我们真正想要捕获的异常是什么呢？最明显的一个是SQLException，这是JDBC操作中常见的异常。另一个可能的异常是IOException，因为它要操作OutputStreamWriter。显然，在同一个catch块中处理这两种截然不同的异常是不合适的。如果用两个catch块分别捕获SQLException和IOException就要好多了。这就是说，catch语句应当尽量指定具体的异常类型，而不应该指定涵盖范围太广的Exception类。

另一方面，除了这两个特定的异常，还有其他许多异常也可能出现。例如，如果由于某种原因，executeQuery返回了null，该怎么办？答案是让它们继续抛出，即不必捕获也不必处理。实际上，我们不能也不应该去捕获可能出现的所有异常，程序的其他地方还有捕获异常的机会??直至最后由JVM处理。

结论二：在catch语句中尽可能指定具体的异常类型，必要时使用多个catch。不要试图处理所有可能出现的异常。

反例之三：占用资源不释放

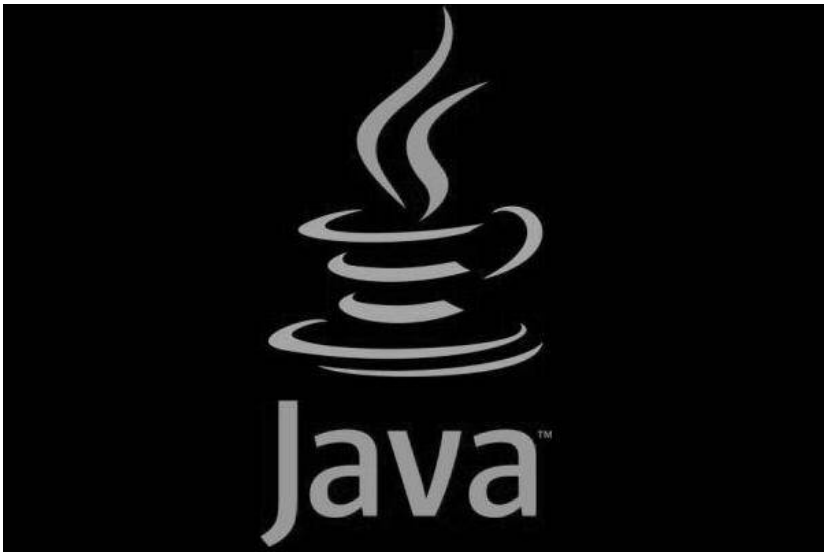
代码：3行-14行。

异常改变了程序正常的执行流程。这个道理虽然简单，却常常被人们忽视。如果程序用到了文件、Socket、JDBC连接之类的资源，即使遇到了异常，也要正确释放占用的资源。为此，Java提供了一个简化这类操作的关键词finally。

finally是样好东西：不管是否出现了异常，Finally保证在try/catch/finally块结束之前，执行清理任务的代码总是有机会执行。遗憾的是有些人却不习惯使用finally。

当然，编写finally块应当多加小心，特别是要注意在finally块之内抛出的异常??这是执行清理任务的最后机会，尽量不要再有难以处理的错误。

结论三：保证所有资源都被正确释放。充分运用finally关键词。



反例之四：不说明异常的详细信息

代码：3行-18行。

仔细观察这段代码：如果循环内部出现了异常，会发生什么事情？我们可以得到足够的信息判断循环内部出错的原因吗？不能。我们只能知道当前正在处理的类发生了某种错误，但却不能获得任何信息判断导致当前错误的原因。

printStackTrace的堆栈跟踪功能显示出程序运行到当前类的执行流程，但只提供了一些最基本的信息，未能说明实际导致错误的原因，同时也不易解读。

因此，在出现异常时，最好能够提供一些文字信息，例如当前正在执行的类、方法和其他状态信息，包括以一种更适合阅读的方式整理和组织printStackTrace提供的信息。

结论四：在异常处理模块中提供适量的错误原因信息，组织错误信息使其易于理解和阅读。

反例之五：过于庞大的try块

代码：3行-14行。

经常可以看到有人把大量的代码放入单个try块，实际上这不是好习惯。这种现象之所以常见，原因就在于有些人图省事，不愿花时间分析一大块代码中哪几行代码会抛出异常、异常的具体类型是什么。把大量的语句装入单个巨大的try块就象是出门旅游时把所有日常用品塞入一个大箱子，虽然东西是带上了，但要找出来可不容易。

一些新手常常把大量的代码放入单个try块，然后再在catch语句中声明Exception，而不是分离各个可能出现异常的段落并分别捕获其异常。这种做法为分析程序抛出异常的原因带来了困难，因为一大段代码中有太多的地方可能抛出Exception。

结论五：尽量减小try块的体积。

反例之六：输出数据不完整

代码：7行-11行。

不完整的数据是Java程序的隐形杀手。仔细观察这段代码，考虑一下如果循环的中间抛出了异常，会发生什么事情。循环的执行当然是要被打断的，其次，catch块会执行??就这些，再也没有其他动作了。已经输出的数据怎么办？使用这些数据的人或设备将收到一份不完整的（因而也是错误的）数据，却得不到任何有关这份数据是否完整的提示。对于有些系统来说，数据不完整可能比系统停止运行带来更大的损失。

较为理想的处置办法是向输出设备写一些信息，声明数据的不完整性；另一种可能有效的办法是，先缓冲要输出的数据，准备好全部数据之后再一次性输出。

结论六：全面考虑可能出现的异常以及这些异常对执行流程的影响。

改写后的代码

根据上面的讨论，下面给出改写后的代码。也许有人会说它稍微有点啰嗦，但是它有了比较完备的异常处理机制。

```
1  OutputStreamWriter out = ...
2  java.sql.Connection conn = ...
3  try {
4    Statement stat = conn.createStatement();
5    ResultSet rs = stat.executeQuery(
6      "select uid, name from user");
7    while (rs.next())
8    {
9      out.println("ID: " + rs.getString("uid") + ", 姓名: " + rs.getString("name"));
10   }
11 }
```

```
12  catch(SQLException sqlex)
13  {
14    out.println("警告：数据不完整");
15    throw new ApplicationException("读取数据时出现SQL错误", sqlex);
16  }
17  catch(IOException ioex)
18  {
19    throw new ApplicationException("写入数据时出现IO错误", ioex);
20  }
21  finally
22  {
23    if (conn != null) {
24      try {
25        conn.close();
26      }
27      catch(SQLException sqlex2)
28      {
29        System.err(this.getClass().getName() + ".mymethod - 不能关闭数据库连接: " + sqlex2.toString())
30      }
31    }
32
33    if (out != null) {
34      try {
35        out.close();
36      }
37      catch(IOException ioex2)
38      {
39        System.err(this.getClass().getName() + ".mymethod - 不能关闭输出文件" + ioex2.toString());
40      }
41    }
42  }
```

本文的结论不是放之四海皆准的教条，有时常识和经验才是最好的老师。如果你对自己的做法没有百分之百的信心，务必加上详细、全面的注释。

另一方面，不要笑话这些错误，不妨问问你自己是否真地彻底摆脱了这些坏习惯。即使最有经验的程序员偶尔也会误入歧途，原因很简单，因为它们确确实实带来了“方便”。所有这些反例都可以看作Java编程世界的恶魔，它们美丽动人，无孔不入，时刻诱惑着你。也许有人会认为这些都属于鸡皮蒜毛的小事，不足挂齿，但请记住：勿以恶小而为之，勿以善小而不为。

相关文章

- 异常处理的最佳实践
- Java的内存回收机制
- Eric Raymond对于几大开发语言的评价
- 一道面试题看 HashMap 的存储方式
- 处理 JavaScript 异常的一个想法
- 异常的代价
- Java编程提高性能时需注意的地方
- Java 日志管理最佳实践
- 我是工程师，不是编译器
- Java程序员集合框架面试题

可能感兴趣的话题

- 2015小米校园招聘笔试题(回文串问题) • 6
- 2015华硕"硕士生"笔试题(瓶子问题) • 14
- 90后在北京买不了房，将何去何从 • 65
- 前辈们带我飞向程序媛 • 52
- 2016美团校园c++工程师校园招聘题
- 毕业工作后，你获取新知识新技能的方式有哪些？ • 1
- 程序猿周末都干什么？ • 120
- 如何解决安卓模拟器启动慢的问题？ • 1
- 祝所有媛们节日快乐~ • 3
- 大家对自由职业怎么看，有成为自由职业者的想法或途径吗？ • 5

« 45个实用的JavaScript技巧、窍门和最佳实践
关于Git的主要维护者滨野纯的访谈 »

登录后评论 新用户注册

直接登录

最新评论

_Kiraa 2012/11/15

修改后的代码看起来太丑陋,丑陋至极.

赞 回复

lzjun 2012/11/15

从现象看本质

赞 回复

hal90 2012/11/16

挺丑的。try本来就是为了解决这种丑代码而设计的，现在又被楼主搞回去了。另外，楼主说要在catch中处理异常，你倒是处理呀！！我还等着看你能在catch里纠正ThreadException呢。 （呲牙）

赞 回复

徐浣泽 2012/11/16

不同意第五条，而且作者其实也没有做任何的修改。引入异常的一个原因就是为了让代码流程更加清晰，如果异常抛出的代码之后没有必要继续执行，那么一个大的try块反而是一个好的办法。对于最后的finally块，可以对关闭的代码做一些封装，例如IOUtils.closeQuitely(Closable)。因为在finally那里如果再无法关闭，除了记录错误信息没有太多可以做的，所以处理逻辑基本上都是一样的。

赞 回复

mark 2013/11/13

实际中，这种写法，是要写死人的.
代码的逻辑不复杂，全部在处理异常，太丑陋了

赞 回复

Edward Shen 2013/11/22

修改后的代码傻极了！
另外，代码怎么处理异常，要根据需要来定。
如果代码是一个网络服务器处理请求的代码。请问可以抛出无法处理的异常吗？如果是，那么一个错误的请求就会造成服务器进程死掉。 显然，应该捕获所有异常，并把栈信息打印到log中。
像数据库close这种操作，直接用Exception捕获然后打印。处理什么异常？ 数据库关不掉，你能怎么样？用户能怎样？ 程序员有空自己看log。没空就算了！
Java世界有一个极大的误区，就是抛出异常的滥用！你close失败，返回个null不就可以了？抛个异常，逼用户去处理，你说傻不傻？！

赞 回复

dk 2014/10/06

修正后的代码 可读性差到死，异常一个也没处理，还抛出了，
这样干嘛还要try catch，

理论派么？

我不信哪个商业项目里像这样写。

赞 回复

业界热点资讯

更多 »

人机大战前 我们采访了那位输给机器的围棋冠军
19 小时前 • 3

微软是真爱上 Linux：SQL Server 也要出 Linux...
19 小时前 • 3

“不耗电”WiFi技术诞生：手机续航有救了
23 小时前 • 3

2015年第四季度全球闪存行业营收同比减少5%
19 小时前 • 2

让 @ 风靡全球的工程师去世了，RIP
2 天前 • 4

精选工具资源

更多资源 »

im4java：ImageMagick命令行Java接口
图像处理

Thumbnailator：高质量缩略图Java开发库
图像处理

AudioKit：强大的iOS音频合成、处理分析工具集
iOS, 音频

EZAudio：iOS/OSX音频可视化框架
iOS, 音频

LingPipe：自然语言处理工具包
自然语言处理

最新评论

- 
  Re：别学框架，学架构
  框架限制了程序员的独立思考能力，别拿规范说事，真正的程序员必须有规范编码的基本功，而不是靠强制的框架...
- 
  Re：女生节的一个分号，引发程序员的...
  果然，评论才是本体哇~ :+1:
- 
  Re：数据库设计规范化的 5 个要求
  “三种独立的表，分别为图书基本信息表、作者基本信息表、图书与作者对应表等等”2张表有什么缺点吗？一张...
- 
  Re：神解释：向外行介绍程序员工作的...
  最后的图。
- 
  Re：在淘宝上买件东西，背后发生了什...
  淘宝的并发量感觉应该是全球第一了
- 
  Re：用 Redis 实现分布式锁与实现...
  有问题，在Redis分布式锁里，$ttl = $this->redisString->t...
- 
  Re：三个月不工作，我才转行做了程序员

梦到的那条蟒蛇，其实是想让你去学Python，然而你却选择了JAVA。。

- Re: 在淘宝上买件东西，背后发生了什...
  先给您的文章点个赞~taobao确实牛啊~系统巨复杂!还有您在文中没有讲到后端很多更复杂和核心的系统...

关于伯乐在线博客

在这个信息爆炸的时代，人们已然被大量、快速并且简短的信息所包围。然而，我们相信：过多“快餐”式的阅读只会令人“虚胖”，缺乏实质的内涵。伯乐在线博客团队正试图以我们微薄的力量，把优秀的原创/译文分享给读者，做一个小而精的精选博客，为“快餐”添加一些“营养”元素。

快速链接
问题反馈与求助 »
加入伯乐翻译小组 »
加入伯乐原创写作 »

关注我们

新浪微博：@伯乐在线官方微博
RSS：订阅地址
推荐微信号

合作联系
Email：bd@Jobbole.com
QQ： 2302462408 （加好友请注明来意）

更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子
头条 - 分享和发现有价值的内容与观点
相亲 - 为IT单身男女服务的征婚传播平台
资源 - 优秀的工具资源导航
翻译 - 翻译传播优秀的外文文章
文章 - 国内外的精选文章
设计 - UI,网页，交互和用户体验
iOS - 专注iOS技术分享
安卓 - 专注Android技术分享
前端 - JavaScript, HTML5, CSS
Java - 专注Java技术分享
Python - 专注Python技术分享

 首页 博客 资源 小组 相亲 反馈 本站由 UCloud 赞助云主机