

Java 异常处理的误区和经验总结

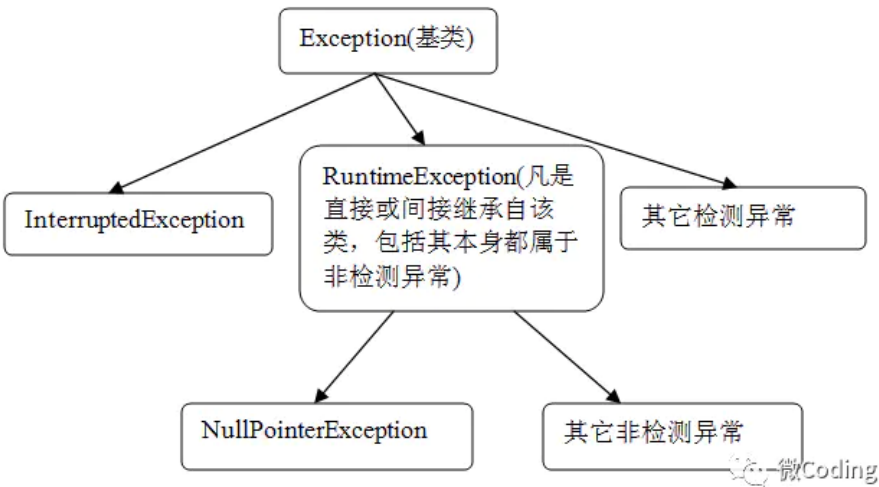
 L千年老妖 已关注

 0.173 2018.12.01 16:47:38 字数 2,562 阅读 42

本文着重介绍了 Java 异常选择和使用中的一些误区，希望各位读者能够熟练掌握异常处理的一些注意点和原则，注意总结和归纳。只有处理好了异常，才能提升开发人员的基本素养，提高系统的健壮性，提升用户体验，提高产品的价值。

误区一、异常的选择

图 1. 异常分类



image

图 1 描述了异常的结构，其实我们都知道异常分检测异常和非检测异常，但是在实际中又混淆了这两种异常的应用。由于非检测异常使用方便，很多开发人员就认为检测异常没什么用处。其实异常的应用情景可以概括为以下：

- 一、调用代码不能继续执行，需要立即终止。出现这种情况的可能性太多太多，例如服务器连接不上、参数不正确等。这些时候都适用非检测异常，不需要调用代码的显式捕捉和处理，而且代码简洁明了。
- 二、调用代码需要进一步处理和恢复。假如将 `SQLException` 定义为非检测异常，这样操作数据时开发人员理所当然的认为 `SQLException` 不需要调用代码的显式捕捉和处理，进而会导致严重的 `Connection` 不关闭、`Transaction` 不回滚、`DB` 中出现脏数据等情况，正因为 `SQLException` 定义为检测异常，才会驱使开发人员去显式捕捉，并且在代码产生异常后清理资源。当然清理资源后，可以继续抛出非检测异常，阻止程序的执行。根据观察和理解，检测异常大多可以应用于工具类中。

误区二、将异常直接显示在页面或客户端。

将异常直接打印在客户端的例子屡见不鲜，以 JSP 为例，一旦代码运行出现异常，默认情况下容器将异常堆栈信息直接打印在页面上。其实从客户角度来说，任何异常都没有实际意义，绝大多数的客户也根本看不懂异常信息，软件开发也要尽量避免将异常直接呈现给用户。

清单 1

```
1 package com.ibm.dw.sample.exception;
2 /**
3  * 自定义 RuntimeException
4  * 添加错误代码属性
5  */
6 public class RuntimeException extends java.lang.RuntimeException {
7     //默认错误代码
8     public static final Integer GENERIC = 1000000;
9     //错误代码
10    private Integer errorCode;
11    public RuntimeException(Integer errorCode, Throwable cause) {
12        this(errorCode, null, cause);
13    }
14    public RuntimeException(String message, Throwable cause) {
15        //利用通用错误代码
16        this(GENERIC, message, cause);
17    }
18    public RuntimeException(Integer errorCode, String message, Throwable cause) {
19        super(message, cause);
20        this.errorCode = errorCode;
21    }
22    public Integer getErrorCode() {
23        return errorCode;
24    }
25 }
```

正如示例代码所示，在异常中引入错误代码，一旦出现异常，我们只要将异常的错误代码呈现给用户，或者将错误代码转换成更通俗易懂的提示。其实这里的错误代码还包含另外一个功能，开发人员亦可以根据错误代码准确的知道了发生了什么类型异常。

误区三、对代码层次结构的污染

我们经常将代码分 Service、Business Logic、DAO 等不同的层次结构，DAO 层中会包含抛出异常的方法，如清单 2 所示：

清单 2

```
1 public Customer retrieveCustomerById(Long id) throw SQLException {
2     //根据 ID 查询数据库
3 }
```

上面这段代码咋一看没什么问题，但是从设计耦合角度仔细考虑一下，这里的 SQLException 污染到了上层调用代码，调用层需要显式的利用 try-catch 捕捉，或者向更上层进一步抛出。根据设计隔离原则，我们可以适当修改成：

清单 3

```
1 public Customer retrieveCustomerById(Long id) {
2     try{
3         //根据 ID 查询数据库
4     }catch(SQLException e){
5         //利用非检测异常封装检测异常，降低层次耦合
6         throw new RuntimeException(SQLErrorCode, e);
7     }finally{
8         //关闭连接，清理资源
9     }
```

```
10 |     }  
    | }
```

误区四、忽略异常

如下异常处理只是将异常输出到控制台，没有任何意义。而且这里出现了异常并没有中断程序，进而调用代码继续执行，导致更多的异常。

清单 4

```
1 | public void retrieveObjectById(Long id){  
2 |     try{  
3 |         //..some code that throws SQLException  
4 |     }catch(SQLException ex){  
5 |         /**  
6 |         * 了解的人都知道，这里的异常打印毫无意义，仅仅是将错误堆栈输出到控制台。  
7 |         * 而在 Production 环境中，需要将错误堆栈输出到日志。  
8 |         * 而且这里 catch 处理之后程序继续执行，会导致进一步的问题*/  
9 |  
10 |         ex.printStackTrace();  
11 |     }  
12 | }
```

可以重构成：

清单 5

```
1 | public void retrieveObjectById(Long id){  
2 |     try{  
3 |         //..some code that throws SQLException  
4 |     }  
5 |     catch(SQLException ex){  
6 |         throw new RuntimeException("Exception in retrieveObjectById", ex);  
7 |     }  
8 |     finally{  
9 |         //clean up resultset, statement, connection etc  
10 |     }  
11 | }
```

这个误区比较基本，一般情况下都不会犯此低级错误.

误区五、将异常包含在循环语句块中

如下代码所示，异常包含在 for 循环语句块中。

清单 6

```
1 | for(int i=0; i<100; i++){  
2 |     try{  
3 |     }catch(XXXException e){  
4 |         //...  
5 |     }  
6 | }
```

我们都知道异常处理占用系统资源。一看，大家都认为不会犯这样的错误。换个角度，类 A 中执行了一段循环，循环中调用了 B 类的方法，B 类中被调用的方法却又包含 try-catch 这样的语句块。褪去类的层次结构，代码和上面如出一辙。

误区六、利用 Exception 捕捉所有潜在的异常

一段方法执行过程中抛出了几个不同类型的异常，为了代码简洁，利用基类 `Exception` 捕捉所有潜在的异常，如下例所示：

清单 7

```
1 public void retrieveObjectById(Long id){
2     try{
3         //...抛出 IOException 的代码调用
4         //...抛出 SQLException 的代码调用
5     }catch(Exception e){
6         //这里利用基类 Exception 捕捉的所有潜在的异常，如果多个层次这样捕捉，会丢失原始异常的有效信息
7         throw new RuntimeException("Exception in retrieveObjectById", e);
8     }
9 }
```

可以重构成

清单 8

```
1 public void retrieveObjectById(Long id){
2     try{
3         //..some code that throws RuntimeException, IOException, SQLException
4     }catch(IOException e){
5         //仅仅捕捉 IOException
6         throw new RuntimeException("指定这里 IOException 对应的错误代码"/code, "Exception
7     }catch(SQLException e){
8         //仅仅捕捉 SQLException
9         throw new RuntimeException("指定这里 SQLException 对应的错误代码"/code, "Exception
10    }
11 }
```

误区七、多层次封装抛出非检测异常

如果我们一直坚持不同类型的异常一定用不同的捕捉语句，那大部分例子可以绕过这一节了。但是如果仅仅一段代码调用会抛出一种以上的异常时，很多时候没有必要每个不同类型的 `Exception` 写一段 `catch` 语句，对于开发来说，任何一种异常都足够说明了程序的具体问题。

清单 9

```
1 try{
2     //可能抛出 RuntimeException、IOException 或者其它；
3     //注意这里和误区六的区别，这里是一段代码抛出多种异常。以上是多段代码，各自抛出不同的异常
4 }catch(Exception e){
5     //一如既往的将 Exception 转换成 RuntimeException，但是这里的 e 其实是 RuntimeException 的
6     throw new RuntimeException("/**/code, /**/, e);
7 }
```

如果我们如上例所示，将所有的 `Exception` 再转换成 `RuntimeException`，那么当 `Exception` 的类型已经是 `RuntimeException` 时，我们又做了一次封装。将 `RuntimeException` 又重新封装了一次，进而丢失了原有的 `RuntimeException` 携带的有效信息。

解决办法是我们可以再 `RuntimeException` 类中添加相关的检查，确认参数 `Throwable` 不是 `RuntimeException` 的实例。如果是，将拷贝相应的属性到新建的实例上。或者用不同的 `catch` 语句块捕捉 `RuntimeException` 和其它的 `Exception`。个人偏好方式一，好处不言而喻。

误区八、多层次打印异常

我们先看一下下面的例子，定义了 2 个类 A 和 B。其中 A 类中调用了 B 类的代码，并且 A 类和 B 类中都捕捉打印了异常。

清单 10

```
1 public class A {
2     private static Logger logger = LoggerFactory.getLogger(A.class);
3     public void process(){
4         try{
5             //实例化 B 类，可以换成其它注入等方式
6             B b = new B();
7             b.process();
8             //other code might cause exception
9         } catch(XXXException e){
10             //如果 B 类 process 方法抛出异常，异常会在 B 类中被打印，在这里也会被打印，从而会打印 2 次
11             logger.error(e);
12             throw new RuntimeException(/* 错误代码 */ errorCode, /*异常信息*/msg, e);
13         }
14     }
15 }
16 public class B{
17     private static Logger logger = LoggerFactory.getLogger(B.class);
18     public void process(){
19         try{
20             //可能抛出异常的代码
21         }
22         catch(XXXException e){
23             logger.error(e);
24             throw new RuntimeException(/* 错误代码 */ errorCode, /*异常信息*/msg, e);
25         }
26     }
27 }
```

同一段异常会被打印 2 次。如果层次再复杂一点，不去考虑打印日志消耗的系统性能，仅仅在异常日志中去定位异常具体的问题已经够头疼的了。

其实打印日志只需要在代码的最外层捕捉打印就可以了，异常打印也可以写成 AOP，织入到框架的最外层。

误区九、异常包含的信息不能充分定位问题

异常不仅要能够让开发人员知道哪里出了问题，更多时候开发人员还需要知道是什么原因导致的问题，我们知道 java.lang.Exception 有字符串类型参数的构造方法，这个字符串可以自定义成通俗易懂的提示信息。

简单的自定义信息开发人员只能知道哪里出现了异常，但是很多的情况下，开发人员更需要知道是什么参数导致了这样的异常。这个时候我们就需要将方法调用的参数信息追加到自定义信息中。下例只列举了一个参数的情况，多个参数的情况下，可以单独写一个工具类组织这样的字符串。

清单 11

```
1 public void retrieveObjectById(Long id){
2     try{
3         //..some code that throws SQLException
4     }catch(SQLException ex){
5         //将参数信息添加到异常信息中
6         throw new RuntimeException("Exception in retrieveObjectById with Object Id :"+
7             id, ex);
8     }
9 }
```

误区十、不能预知潜在的异常

在写代码的过程中，由于对调用代码缺乏深层次的了解，不能准确判断是否调用的代码会产生异常，因而忽略处理。在产生了 Production Bug 之后才想起来应该在某段代码处添加异常捕捉，甚至不能准确指出出现异常的原因。这就需要开发人员不仅知道自己在做什么，而且要去尽可能的知道别人做了什么，可能会导致什么结果，从全局去考虑整个应用程序的处理过程。这些思想会影响我们对代码的编写和处理。

误区十一、混用多种第三方日志库

现如今 Java 第三方日志库的种类越来越多，一个大项目中会引入各种各样的框架，而这些框架又会依赖不同的日志库的实现。最麻烦的问题倒不是引入所有需要的这些日志库，问题在于引入的这些日志库之间本身不兼容。如果在项目初期可能还好解决，可以把所有代码中的日志库根据需要重新引入一遍，或者换一套框架。但这样的成本不是每个项目都承受的起的，而且越是随着项目的进行，这种风险就越大。

怎么样才能有效的避免类似的问题发生呢，现在的大多数框架已经考虑到了类似的问题，可以通过配置 Properties 或 xml 文件、参数或者运行时扫描 Lib 库中的日志实现类，真正在应用程序运行时才确定具体应用哪个特定的日志库。

其实根据不需要多层次打印日志那条原则，我们就可以简化很多原本调用日志打印代码的类。很多情况下，我们可以利用拦截器或者过滤器实现日志的打印，降低代码维护、迁移的成本。

结束语

以上纯属个人的经验和总结，事物都是辩证的，没有绝对的原则，适合自己的才是最有效的原则。希望以上的讲解和分析可以对您有所帮助。

 0人点赞 >



 Java基础



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



L千年老妖 编程爱好者
总资产32 (约2.28元) 共写了3.7W字 获得8个赞 共8个粉丝

已关注



试题题库



自己学编程



网店购买



车险报价



滚筒输送机



包装箱



写下你的评论...

全部评论 0

只看作者

按时间倒序

按时间正序

被以下专题收入，发现更多相似内容

+ 收入我的专题

推荐阅读

更多精彩内容>

java异常处理的经验总结

这篇文章主要是对Java异常选择和使用中的一些误区的总结和归纳，希望各位读者能够熟练掌握异常处理的一些注意点和原则...

 唐老鸭z 阅读 221 评论 0 赞 0



Java异常处理的11大误区及经验总结

在写代码的过程中，我们往往会忽略一些异常处理的基础知识。本文将着重介绍Java 异常选择和使用中的一些误区，希望...

 小宇java 阅读 213 评论 0 赞 4




猿学－玩转Java自定义异常的教程

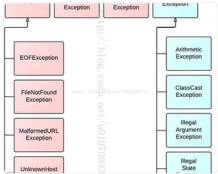
package testexceptiom; import java.text.ParseException; im...

 猿学 阅读 435 评论 0 赞 1

Java学习总结之异常处理


引言 在程序运行过程中(注意是运行阶段，程序可以通过编译)，如果JVM检测出一个不可能执行的操作，就会出现运行时错...

 Steven1997 阅读 1,053 评论 1 赞 6



跳舞的青春

也许，那份勇敢还是没有来，让我独自一人在自己的地盘嚣张的不知所措，让我害怕的在生活里活得不成样子。许多错误的发生...

 未暖时光 阅读 25 评论 0 赞 0

