Java 中可检查异常的使用策略

STRATEGIES OF USING CHECKED EXCEPTION IN JAVA

肖 连 (西安理工大学 计算机科学与工程学院 西安 710048)
(郑州航空工业管理学院计算机系 郑州 450015)
崔杜武 (西安理工大学 计算机科学与工程学院 西安 710048)

摘要:强大的异常处理是 Java 的一大优势。可检查异常是程序设计中流程所能控制的部分,也是程序设计中异常处理的核心。有效的使用可检查异常能加强程序的健壮性。本文对于可检查异常的处理提出一些使用策略。
关键词:Java 可检查异常 策略

1 引言

以前的语言采用返回值的方法来判断程序运行的状况,这种方法的主要缺点是:将判断语句和主程序放在一起,造成主程序代码的庞大和复杂;而且返回值没有特别的意义,程序员可能根本不去验证返回值,时间长久了代码的可读性,易维护性变差。而采用异常处理就可以避免以上缺陷。强大的异常处理是 Java 的一大优势[1]。使用异常处理可以将异常处理过程和主程序代码分开,同时也加强了程序的易维护性。Java 提供了大量的异常类,且对于可检查异常要求强制声明或捕获。本文探讨怎样合理有效的利用 Java 的异常处理。

2 异常的分类

Java 中的内置异常类的分类情况如图 1 所示。

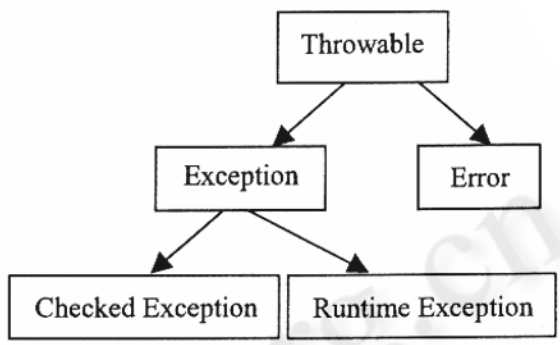

图 1 异常分类

Java 异常分为 Exception 和 Error 两类[3][4],它们都是 Throwable 的子类。Error 对应 0 的是系统级的错误。例如磁盘坏道,操作系统损毁,硬盘故障,CPU 故障等。这类异常是程序所不能恢复的,此种状况一般称为错误。

Checked Exception 和 Runtime Exception 都是 java.lang.exception 的子类。

Runtime Exceptiom 又称为 Unchecked Exception。Unchecked Exception 表示编译器不会检查程序是否对 Runtime Exception 作了处理,在程序中不必捕获 Runtime Exception 类型的异常,也不必在方法体声明抛出 Runtime Exception 类。如 ArithmeticException, NullPointerException, IndexOutOfBoundsException。Runtime Exception 发生时表示程序中出现了编程错误,所以应该找出错误修改程序,而不是去捕获和处理 Runtime Exception。

Checked Exception 是在编程中使用最多的异常类,所有继承自 Exception 并且不是 Runtime Exception 的异常都是 Checked Exception。例如 ClassNotFoundException, IOException。Java 语言规定必须对 Checked Exception 作处理,编译器会对此作检查,要么在方法体中声明抛出 Checked Exception,要么使用 catch 语句捕获 Checked Exception 进行处理,不然不能通过编译,否则会产生 unre-

ported exception yourClassName; must be caught or declared to be thrown 错误。

3 异常处理机制

Java 语言的异常处理采用 try,catch,finally,throw,throws 五个保留字来进行描述。try-catch-finally 语句组用于捕获处理异常;throw 用于自主抛出异常,抛出异常后由被调用方法进行捕捉,若没有相应的类型匹配,则交由调用方法进行处理,若调用方法也没有相应的类型匹配,则一直向上面传递,若在这个过程中没有方法处理该异常,那么程序将终止;throws 关键字用于在方法声明时来说明可能会抛出的异常。若一个异常在被调用方法中有可能产生,那么在调用方法中就必须进行处理,此时,一般在定义方法时用 throws 关键字来进行说明。

Java 要求对于可检查异常必须进行声明或处理,所以在编写方法之前,应阅读 API 的联机文档,掌握方法内所需要声明或处理的异常。

4 可检查异常的使用策略

对于 Error 类,采取的方式就是终止程序运行,修复系统级的错误。发生 Runtime 类的异常,程序也自动的中断,由程序员来进行修改程序的编码,改正此类的错误。而对于可检查异常,是可恢复的异常状态,遇见此类异常,系统有望重新运行,直到得到一个正确的状态为止。所以程序员对此类异常进行处理和控制。可检查异常的使用原则如下所述。

4.1 不要丢弃异常

在 Java 中存在关闭异常的强烈倾向,如果编写代码调用某方法,预测该方法不会抛出异常,一般人不会在该方法的 throwable 列表中说明该异常。若不说明异常,编译器将会报错。所以一般人只好关闭异常:

```
try{
  //业务处理代码
} catch (Exception e) {
}
```

对于 Java 语言使用不熟练的程序员来说,他们有时为了程序的运行也会编写如此代码。程序中如发生异常则什么也不做,这样程序若有异常发生就得不到任何信息。

4.2 不要过分细化异常

Java 是通过堆栈来处理异常的,JVM 在执行异常处理是采用反复的在堆栈中向下匹配的方式直到到达 ThreadGroup. uncauthtException( )为止,因此整个过程需要付出很大的代价;另一方面,即使异常没有被捕获,处理包含异常的代码仍需要较多的运行时间[2]。

下面的代码是不可取的

```
//不可取代码
for (i=0; i<100; i++) {
 try{
  //主代码
 } catch (SuchException e) {
  //异常处理代码
}
}
```

此循环体正常情况下执行一百次,而在每次都捕获异常就过分细化了异常,导致异常代码迅速膨胀。好的程序员应对代码作如下更改:

```
//可取代码
try{
 for (i=0; i<100; i++) {
  //主代码
 }
}catch (SuchException e) {
  //异常处理代码
}
```

4.3 注意异常的层次关系

当一个 try…catch 语句中含有多个 catch 语句块时,我们要注意各 catch 语句块的异常参数的类型,基本原则是不能够有多个同一种类型的异常参数,否则会有编译的错误,原因是异常在最初的那个 catch 块就已经被捕获了。在一个程序中可以抛出多个异常。有父子继承关系的异常类若用于同一个 try…catch 语句里,则捕捉子类的 catch 语句块要在前

面,而捕捉父类的 catch 语句块要在后面,否则子类异常对象产生时,先遇到父类的 catch 语句块就已经被捕捉了,其后子类的 catch 语句块就变成多余的,则编译时也会产生 exception 子类 has been caught 的错误。

```
try{
//主代码
}catch(SubSubSub1Exception e){
//处理
}catch(SubSubException e){
//处理
}catch(SubException e){
//处理
}catch(Exception e){
//处理
}
```

4.4 捕获并处理异常

对于可检查异常来说,捕获只是第一步,只输出异常信息是不够的。显示给用户的异常信息应转换为易于理解的信息。好的程序应该能处理异常,使程序能继续运行下去。下面给出一个简单的例子:由用户输入文件名,若文件存在,读取文件的程序。若输入错误,程序不应该中断,而是通知用户,错在哪里,然后返回重新输入,直到输入正确为止。代码如下:

```
do {
  //输入文件名
 try{
  //读取文件
}catch (SuchException e) {
 //提示用户错误
}
} while(文件不存在)
```

4.5 finally 块不要抛出异常

finally 块一般用于资源的释放,例如关闭链接等。[2] finally 块是必须执行的块。若 try 块内程序没有发生异常,则不执行 catch 块,然后执行 finally 块,若 try 块内程序发生异常,则在发生处中断查找下面的 catch 块进行异常类型匹配,然后执行相应的块,接着执行 finally 块。若在 finally 块中抛出异常,此异常将覆盖 try 块中的异常,导致主代码中的异常不能显示,这是程序所不希望的。所以,finally 块中一般利用简单的测试机制。例如,关闭数据库链接的语句,就可以加上一个 if 条件判断:

```
try{
//打开数据库
}catch (suchException e) {
//捕获处理
}finally {
if (con! =null)
{
con.close();
}
}
```

5 总结

Java 中的异常处理是高级语言中比较先进和全面的。合理有效的利用 Java 异常处理机制特别是对可检查异常的处理,可以加强程序的健壮性,可移植性和可维护性。

参考文献

1 赵化冰、唐英、唐文彬、芦东昕,Java 异常处理,计算机应用(J),2003,12,第 23 卷.

2 江义华,Java 完美经典,中国铁道出版社,2004,3,第一版.

3 (美)Joshua Bloch 著 潘爱民,Effective Java 中文版,机械工业出版社 2003.1.

4 (美)James Gosling,Bill Joy,Guy Steele 著,蒋国新,朱暄,李洪伟 译,Java 语言规范,北京大学出版社,1997 年 12 月第一版.