异常处理机制的陷阱

本笔记知识点来自于李刚《Java程序员的基本修养》

- finally块
- catch块

1: finally块

在实际的开发过程中，我们需要对很多的物理资源，比如说数据库，磁盘文件进行操作。当我们对这些资源进行操作完毕之后，必须由程序员显示的关闭这些物理资源，否则会造成物理资源泄漏。**值得注意的是，JVM的垃圾回收机制是内存管理的一部分，它并不会涉及到物理资源的操作，因此垃圾回收机制不会对物理资源进行任何的操作。**

1. 传统关闭物理资源方式
   记住三要点：

   - 使用finally块来关闭物理资源
   - 关闭物理资源之前，要先判定每个资源的引用变量不为null
   - 为每个物理资源使用单独的try...catch块来进行关闭，以免造成不同物理资源关闭过程中出现的异常不会造成其他物理资源关闭过程
     看下面例子：

   ```
   public class Test {
   public static void main(String[] args) {
       Wolf w1 = new Wolf("hh", 23);
       Wolf w2 = null;
       try
       {
       ObjectOutputStream oos = new ObjectOutputStream(new
   FileOutputStream("a.bin"));
       ObjectInputStream ois = new ObjectInputStream(new
   FileInputStream("b.bin"));
       oos.writeObject(w1);
       oos.flush();
       w2 = (Wolf) ois.readObject();
       }
       finally {
           if (oos != null) {
               try {
                   oos.close();
               }catch (Exception e) {
                   e.printStackTrace();
               }
           }
           if (ois != null) {
               try {
                   ois.close();
               }catch (Exception e) {
                   e.printStackTrace();
               }
           }
       }
   }
   }
   ```

   很臃肿的代码。

2. 增强版try语句关闭资源
   在Java7以后，基本上Java重写了所有的资源类，因此可以有一种简洁的方法来进行物理资源的自动关闭。那就是try语句后面跟在跟花括号之前先跟***一对圆括号，在这对圆括号里面进行对物理资源引用变量的命名和初始化。***这个try语句既没有catch块也没有finally块，因为它会隐式的调用finally块。看下面例子：

   ```
   public class Test {
       public static void main(String[] args) {
           Wolf w1 = new Wolf("hh", 23);
           Wolf w2 = null;
           try (
               ObjectOutputStream oos = new ObjectOutputStream(new
   FileOutputStream("a.bin"));
               ObjectInputStream ois = new ObjectInputStream(new
   FileInputStream("b.bin"));
               //在圆括号里面进行命名和初始化，当物理资源使用完毕之后，会自
   动进行关闭
           )
           {
               oos.writeObject(w1);
               oos.flush();
               w2 = (Wolf) ois.readObject();
           }
       }
   }
   ```

   使用try语句进行物理资源自动关闭的两个注意点：

   - 被自动关闭的资源必须实现Closable和AutoClosable接口。基本上都实现了；
   - 被自动关闭的资源必须在try语句堆圆括号对里面进行命名和初始化

3. finally块的陷阱
   finally块代表总会被执行的代码，除了一种特殊情况。就是线程停止，也就是Java虚拟机退出。 `System.exit(0);` 语句运行时，线程结束，finally块并不会被执行。 `return;` 语句执行之后，线程并没有结束，finally代码块会被执行。
   接下来看几个有趣的例子：

   - case1:

4. ```
   public class Test {
   public static void main(String[] args) {
       int a = test();
       System.out.println(a);
   }
   public static int test() {
       int count = 5;
       try  {
           System.out.println("here1");
           return count++;
       }
       finally {
           System.out.println("here2");
           return ++count;
       }
   }
   }
   ```

   这段代码输出值是`7`,这意味着在`try`语句里面的`return`语句已经执行完毕，因为只有`return`执行完毕才能使得`count++`;但是最终输出结果为`7`,意味着`try`代码块结束后，程序跑到了`finally`代码块中，同时`++count`也执行了，而且在这里返回了，因此`try`代码块中的返回操作不会继续进行下去，只是刚好快要执行就被`finally`中的`return`打断了。

   - case2:

5. 从上面两个例子来看，try语句中的return语句会立马执行完毕，但是在结束方法之前，会去寻找是否存在finally代码块，如果没有，返回数据，结束方法；如果存在，执行finally代码块，如果在finally代码块中存在返回操作，直接在finally中执行返回操作，结束方法，并不会回到try代码块中去返回了。因此可以总结为：

   ```
   public class Test {
   public static void main(String[] args) {
       int a = test();
       System.out.println(a);
   }
   public static int test() {
       int count = 5;
       try  {
           System.out.println("here1");
           return count++;
       }
       finally {
           System.out.println("here2");
           ++count;
       }
   }
   }
   ```

   这段代码输出结果是`5`,和之前一样，`try`代码块准备`return 5`时，发现有`finally`语句块，于是执行`finally`代码块，在这段代码块中`count++`;但是这里并没有返回操作，因此有效对返回数据仍然是`try`代码块中的返回值。

   - 如果try代码块中有return语句，finally中没有，那么按照try代码块中的return语句返回值，就算finally中有对数据对操作也对try代码块返回操作无效，因为try中的返回操作是立即执行的；
   - 如果try代码块中有return语句，finally代码块中也有，那么可以这样理解，try中返回操作和finally返回操作都会执行，只不过finally中的返回操作会覆盖try中的返回操作。

6. catch块的用法
   在Java中，对于非自动关闭资源的try代码块，不能是一个孤零零的代码块，必须配对一个catch代码块或者一个finally代码块。一个try块可以不仅仅只对应一个catch块，还可以对应多个catch代码块。

   1. catch块的顺序：
      catch代码块的顺序和我们之前说过的if...else代码块的实现顺序是一样的。都是从小范围到大范围。因此如果存在多个catch代码块，必须先catch小的异常，然后再逐层catch大的异常；
   2. catch代码块的修复功能
      一般来说，当我们捕捉到一个异常，我们需要对这个异常进行修复。因此修复异常的语句一般存在于catch代码块中。但值得注意的是，修复语句最好不要再存在抛出异常可能性；
   3. catch异常的类型
      凡事都要对症下药，catch异常也是同样的。catch语句捕捉的对象只是代码中可能抛出的异常，而不可能抛出的异常，编译器是不会通过的。
   4. 继承得到的异常
      Java语言规定，子类继承父类所抛出的异常，不能是父类抛出异常的范围更大，数量更多的异常。而只能是父类异常的子类。
   ```