

JAVA狂飙之路1-Exception异常系

2020年12月6日 日曜日 19:34

异常系产生原因

1. 用户输入了非法数据
2. 要打开的文件不存在
3. 网络通信故障或JAVA内存溢出

异常分类

1. 检查性异常
(IOException)
2. 运行时异常
(RuntimeException)
3. 错误(Error)

概论以及异常系分类

1. Error

Java程序一般不捕捉错误Error.

Error用来指示运行时环境的错误.

错误一般发生在严重故障时,它们在java程序处理的范畴之外.

例如: 操作系统崩溃, jvm出错跟内存溢出, 动态链接库失败等.

Error并不是异常,.一般的,程序不会从错误中恢复.

2. 所有的异常都是从java.lang.Exception类继承的子类.

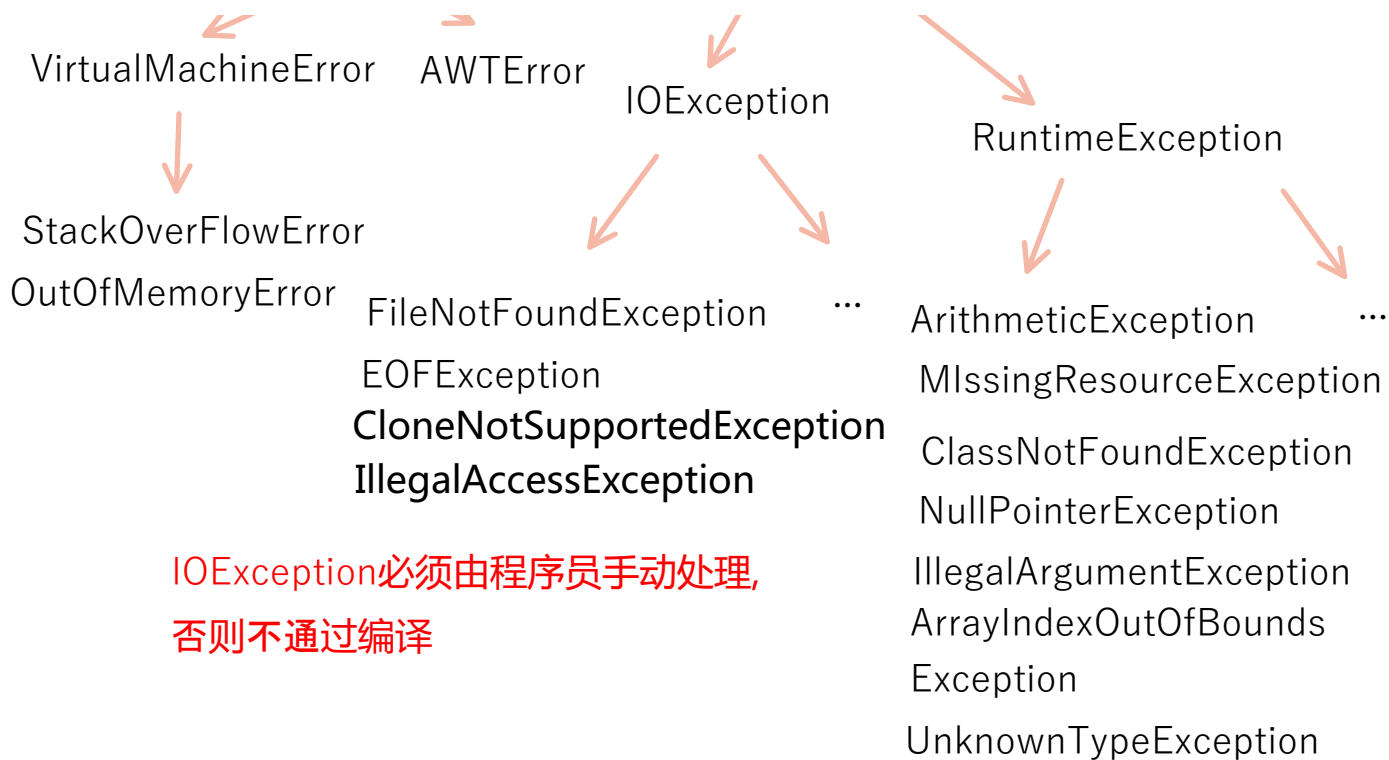
Exception类是Throwable类的子类.

3. Throwable代表是可抛出的,有两个子类: Exception跟Error.

Exception也有来两个子类:

IOException(非运行时异常)类和RuntimeException(运行时异常)类





例子

ArithmeticException

运行时异常的派生类有很多, 其产生频率较高. 它的派生类可以由程序处理或者抛给(throw) 给jvm处理. 例如除数为0,就是抛给了jvm处理, jvm把程序中断执行, 并把错误信息输出到终端上.

★ 异常系处理

1. 可能出现的异常使用try/catch捕捉处理
2. 当前函数并不处理异常,使用throw or throws关键字, 把可能出现的异常抛给调用该函数的上级函数处理

例如:

a()函数调用b()函数,预见b()函数可能已有异常,但是在b()函数里不想处理可能产生的异常,就在b()函数里使用throw
或throws关键字,见异常抛给调用b()函数的a()函数处理,在a()函数里使用try/catch结构捕捉.

3. 交给JVM虚拟机处理

⚠ 只适用于RuntimeException及其子类

4. 项目中不建议抛给JVM处理,尽量手动处理

★ Throws/Throw关键字

1. 如果一个方法没有捕获到一个检查性异常(IOException), 那么该方法必须使用 throws 关键字来声明。throws 关键字放在方法签名的尾部。

即: 如果一个方法里利用throw手动抛出1个非RuntimeException异常, 必须在

函数定义声明里加上throws 后缀

也可以使用 throw 关键字抛出一个异常, 无论它是新实例化的还是刚捕获到的。

```
public void deposit(double amount) throws  
RemoteException  
{  
    // Method implementation  
    throw new RemoteException();  
}
```

2. 一个方法可以声明抛出多个异常, 多个异常之间用逗号隔开。例如, 下面的方法声明抛出

RemoteException 和 InsufficientFundsException :

```
public void withdraw(double amount) throws  
RemoteException,  
InsufficientFundsException  
{  
    // Method implementation  
}
```

3. 语法:

throw new XException();

其中XException必须是Exception的派生类.

这里注意throw 出的是1个异常**对象**, 所以new不能省略
作用就是手动令程序抛出1个异常对象.

★ finally关键字 ➡ try catch finally的执行路线.

下面用个例子来说明:

```

1  try{
2      f();
3      ff();
4  }
5  catch(ArithmeticException e){
6      g();
7  }
8  catch(IOException e){
9      gg();
10 }
11 catch(AuthorizedException e){
12     ggg();
13 }
14 finally{
15     h();
16 }
17
18 k();

```

1. 当try里面的f()抛出了IOException

当f()抛出了异常, 那么ff()就不会执行了. 程序会尝试捕捉异常.

首先捕捉ArithmeticException, 捕捉失败.

接下来捕捉IOException, 捕捉成功, 执行gg();

一旦捕捉到一个异常, **不会再尝试捕捉其他异常**, 直接执行finally里的h();

执行后面的函数k().

也就是说路线是:

f() -> gg() -> h() -> k()

⚠ 有2点要注意的.

1. f()函数极有可能未完整执行, 因为它抛出了异常, 抛出异常的语句执行失败, 之后的语句放弃执行.

2. try{} 里面, f()之后的语句, 例如ff()放弃执行.

2. 没有任何异常抛出

这种情况很简单, 就是try{}里面的代码被完整执行, 因为没有抛出任何异常, 就不会尝试执行catch里的部分, 直接到finally部分了.

路线是:

f() -> ff() -> h() -> k()