链滴

Java 基础知识系列——异常 (旧文搬运)

作者：DeeWooo

原文链接：https://ld246.com/article/1575679118777

来源网站：链滴

什么是异常?

先看一段代码

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int a = scanner.nextInt();
    System.out.println("a = "+a);
}
```

这段代码的执行结果是

```
2
a = 2
```

如果我们输入不是整型数2，而是一个字符串"abc",那么它的执行结果是

```
abc
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:864)
at java.util.Scanner.next(Scanner.java:1485)
at java.util.Scanner.nextInt(Scanner.java:2117)
at java.util.Scanner.nextInt(Scanner.java:2076)
at com.dd299.imserver.ExceptionTest.main(ExceptionTest.java:17)
```

在这里，程序出了错，没有返回我们期望的结果形式，而是打印出一堆错误信息。这种情况，就叫做常。

从上面的例子可以看出，在java中，异常就是程序的一些错误。

异常和哪些错误有关?

1. 用户输入错误
2. 设备错误
3. 物理限制
4. 代码错误

在java中，所有的异常对象都是继承于Throwable类的一个实例。事实上，直接继承自Throwable的只有两个：Exception和Error。

- Error：主要描述系统内部错误和资源耗尽的错误，是脱离程序员控制的问题。
- Exception：是我们常说的异常，程序员可以通过优化代码的方式避免它。异常又分成两类：
    - 运行时异常（RuntimeException）：指在编译期间无法发现的异常，在Java语言规范中，将派于RuntimeException和Error的所有异常称为未检查异常。包含下面几种情况：
        - 错误的类型转换
        - 数组访问越界

- 访问空指针
  - 已检查异常：指编译时就可以发现的异常。包含的情况有：
    - 试图在文件尾部后面读取数据
    - 试图打开一个错误格式的URL
    - 试图根据给定的字符串查找Class对象，而这个字符串表示的类并不存在。

一句永远正确的话： “如果出现了RuntimeException,那一定就是你的问题” 。

处理异常都有哪几种方式

面对出错的情况一向都是两种处理角度，一种就是自己将错误处理掉；另一种是将错误信息抛出去，知调用方，等待处理。相应的，处理异常也一样有两种方式：

- try/catch块：捕获异常
- throws Exception,...:声明已检查异常。

声明已检查异常

对于一个已知有可能抛出异常的方法，需要在方法声明时同时声明可能会抛出的异常

```
public void function() throws Exception{
    exp();
}
publi void exp() throws Exception{
}
```

这种情况，往往是在代码中调用了一个同样声明了异常的方法。

另外对于自定义异常，java不会抛出开发者自定义的异常，这时开发者要自行抛出。

```
public void exp(String input) throws XXXException{
    ...
    if("a".equal.(input)){
        throw new XXXException();
    }
}
```

如何捕获异常?

在上面的例子中，除了将异常抛出去之外，还可以自行处理，这就需要捕获异常。下面的代码说明了个典型的捕获异常的情况

```
public void function() {
    try{
        exp();//(1)
    }
    catch(XXXException xxxe){
    ...//(2)
    }
    catch(Exception e){
    ...//(3)
    }


}
publi void exp() throws Exception{
}
```

上面的例子中，如果exp（）没有出现错误，那么程序执行完try中的代码，就自动结束，不会调用catch中的代码；在try中一旦出现了catch对应的异常，就会停止当前的执行，跳到对应的catch中执行；果出现了异常，但在catch中没找到对应的处理，就退出。

Finally子句

Finally子句是用来处理那些不论有没有发生异常，都要处理的情况，比如说资源的释放。

```
InputStream in = ...;
try{
...
}
finally{
in.close;
}
```