Java声明和抛出异常

# throws声明异常、throw抛出异常、throw和throws的区别

[Java](http://c.biancheng.net/java/) 中的异常处理除了包括捕获异常和处理异常之外，还包括声明异常和拋出异常，可以通过 throws 关键字在方法上声明该方法要拋出的异常，然后在方法内部通过 throw 拋出异常对象。本节详细介绍在 Java 中如何声明异常和拋出异常。  
  
**throws 关键字和 throw 关键字在使用上的几点区别如下**：

* throws 用来声明一个方法可能抛出的所有异常信息，throw 则是指拋出的一个具体的异常类型。
* 通常在一个方法（类）的声明处通过 throws 声明方法（类）可能拋出的异常信息，而在方法（类）内部通过 throw 声明一个具体的异常信息。
* throws 通常不用显示地捕获异常，可由系统自动将所有捕获的异常信息抛给上级方法； throw 则需要用户自己捕获相关的异常，而后再对其进行相关包装，最后将包装后的异常信息抛出。

## throws 声明异常

当一个方法产生一个它不处理的异常时，那么就需要在该方法的头部声明这个异常，以便将该异常传递到方法的外部进行处理。可以使用 throws 关键字在方法的头部声明一个异常，其具体格式如下：

returnType method\_name(paramList) throws Exception 1,Exception2,…{…}

其中，returnType 表示返回值类型，method\_name 表示方法名，Exception 1，Exception2，… 表示异常类。如果有多个异常类，它们之间用逗号分隔。这些异常类可以是方法中调用了可能拋出异常的方法而产生的异常，也可以是方法体中生成并拋出的异常。

#### 例 1

创建一个 readFile() 方法，该方法用于读取文件内容，在读取的过程中可能会产生 IOException 异常，但是在该方法中不做任何的处理，而将可能发生的异常交给调用者处理。在 main() 方法中使用 try catch 捕获异常，并输出异常信息。代码如下：

1. import java.io.FileInputStream;
2. import java.io.IOException;
3. **public** **class** Test04
4. {
5. **public** void readFile() throws IOException
6. {
7. //定义方法时声明异常
8. FileInputStream file=**new** FileInputStream("read.txt"); //创達 FileInputStream 实例对象
9. int f;
10. **while**((f=file.read())!=-1)
11. {
12. System.out.println((char)f);
13. f=file.read();
14. }
15. file.close();
16. }
17. **public** **static** void main(String[] args)
18. {
19. Throws t=**new** Test04();
20. **try**
21. {
22. t.readFile(); //调用 readFHe()方法
23. }
24. **catch**(IOException e)
25. { //捕获异常
26. System.out.println(e);
27. }
28. }
29. }

以上代码，首先在定义 readFile() 方法时用 throws 关键字声明在该方法中可能产生的异常，然后在 main() 方法中调用 readFile() 方法，并使用 catch 语句捕获产生的异常。  
  
注意：在编写类继承代码时要注意，子类在覆盖父类带 throws 子句的方法时，子类的方法声明中的 throws 子句不能出现父类对应方法的 throws 子句中没有的异常类型，因此 throws 子句可以限制子类的行为。也就是说，子类方法拋出的异常不会超过父类定义的范围。

## throw 拋出异常

throw 语句用来直接拋出一个异常，后接一个可拋出的异常类对象，其语法格式如下：

throw ExceptionObject;

其中，ExceptionObject 必须是 Throwable 类或其子类的对象。如果是自定义异常类，也必须是 Throwable 的直接或间接子类。例如，以下语句在编译时将会产生语法错误：

1. **throw** **new** String("拋出异常"); //因为String类不是Throwable类的子类

当 throw 语句执行时，它后面的语句将不执行，此时程序转向调用者程序，寻找与之相匹配的 catch 语句，执行相应的异常处理程序。如果没有找到相匹配的 catch 语句，则再转向上一层的调用程序。这样逐层向上，直到最外层的异常处理程序终止程序并打印出调用栈情况。

#### 例 2

在某仓库管理系统中，要求管理员的用户名需要由 8 位以上的字母或者数字组成，不能含有其他的字符。当长度在 8 位以下时拋出异常，并显示异常信息；当字符含有非字母或者数字时，同样拋出异常，显示异常信息。代码如下：

1. import java.util.Scanner;
2. **public** **class** Test05
3. {
4. **public** boolean validateUserName(String username)
5. {
6. boolean con=**false**;
7. **if**(username.length()>8)
8. { //判断用户名长度是否大于8位
9. **for**(int i=0;i<username.length();i++)
10. {
11. char ch=username.charAt(i); //获取每一位字符
12. **if**((ch>='0'&&ch<='9')||(ch>='a'&&ch<='z')||(ch>='A'&&ch<='Z'))
13. {
14. con=**true**;
15. }
16. **else**
17. {
18. con=**false**;
19. **throw** **new** IllegalArgumentException("用户名只能由字母和数字组成！"");
20. }
21. }
22. }
23. else
24. {
25. throw new IllegalArgumentException("用户名长度必须大于 8 位！");
26. }
27. return con;
28. }
29. public static void main(String[] args)
30. {
31. Test05 te=new Test05();
32. Scanner input=new Scanner(System.in);
33. System.out.println("请输入用户名：");
34. String username=input.next();
35. try
36. {
37. boolean con=te.validateUserName(username);
38. if(con)
39. {
40. System.out.println("用户名输入正确！");
41. }
42. }
43. catch(IllegalArgumentException e)
44. {
45. System.out.println(e);
46. }
47. }
48. }

如上述代码，在 validateUserName() 方法中两处拋出了 IllegalArgumentException 异常，即当用户名字符含有非字母或者数字以及长度不够 8 位时。在 main() 方法中，调用了 validateUserName() 方法，并使用 catch 语句捕获该方法可能拋出的异常。  
  
运行程序，当用户输入的用户名包含非字母或者数字的字符时，程序输出异常信息，如下所示。

1. 请输入用户名：
2. administrator@#
3. java.lang.IllegalArgumentException: 用户名只能由字母和数字组成！

当用户输入的用户名长度不够 8 位时，程序同样会输出异常信息，如下所示。

请输入用户名：

admin

java.lang.IllegalArgumentException: 用户名长度必须大于 8 位！