第9章 异常

2018年7月19日

10:09

**第57条：只针对异常的情况才使用异常**

顾名思义，异常应该只用于异常情况下，它们永远不应该用于正常的控制流。

设计良好的API不应该强迫它的客户端为了正常的控制流而使用异常。

**第58条：对可恢复的情况使用受检异常，对编程错误使用运行时异常**

Java提供了三种可抛出结构：受检的异常，运行时异常和错误。

如果期望调用者能够是当地恢复，对于这种情况就应该使用受检的异常。

用运行时异常来表明编程错误。

实现的所有未受检的抛出接结构都应该RuntimeException的子类。

**第59条：避免不必要地使用受检的异常**

在实战中，catch块几乎总是具有断言失败的特征。

“把受检的异常变成未受检的异常”也是一种避免方法。

**第60条：优先使用标准的异常**

重用现有的异常主要好处是它使你的API更加易于学习和使用，与熟悉的习惯用法是一致的。

如果希望稍微增加更多的失败-捕获信息，可以放心地把现有的异常进行子类化。

**第61条：抛出与抽象相对应的异常**

更高层的实现应该捕获低层的异常，同时抛出可以按照高层抽象进行解释的异常。这种做法被称为异常转译。

可以使用异常链将底层异常（原因）传到高层的异常，高层通过异常的getCause()来获得底层的异常。

异常链对高层和底层异常都提供了最佳的功能：它允许抛出适当的高层异常，同时又能捕获底层的原因进行失败分析。

**第62条：每个方法抛出的异常都要有文档**

始终要单独地声明受检的异常，并且利用Javadoc的@throws标记，准确地记录下抛出每个异常的条件。

使用Javadoc的@throws标签记录下一个方法可能抛出的每个未受检异常，但是不要使用throws关键字

将未受检的异常包含在方法的声明中。

如果一个类中的许多方法出于同样的原因而抛出同一个异常，在该类的文档注释中对这个异常建立文档，这是

可以接受的，而不是为每个方法单独建立文档。

**第63条：在细节消息中包含能捕获失败的信息**

为了捕获失败，异常的细节信息应该包括所有“对该异常有贡献”的参数和域的值。

为异常的“失败捕获”信息提供一些访问方法是合适的

**第64条：努力使失败保持原子性**

失败的方法调用应该使对象保持在被调用之前的状态，具有这种属性的方法具有失败原子性。

保持失败原子性的办法：

* 设计一个不可变对象。
* 调整计算处理的顺序，使得任何可能会失败的计算部分都在对象状态被修改之前发生。
* 编写一段恢复代码来拦截操作过程中发生的失败。

**第65条：不要忽略异常**

不管异常代表了可预见的异常条件还是编程错误，都不要用空的catch块忽略它。

只要将异常传播给外界，至少会导致程序迅速地失败，从而保留了有助于调试该失败条件的信息。