Java理论与实践:关于异常的争论 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/267/2021_2022_Java_E7_90_86_E8_AE_BA_c104_267895.htm 关于在 Java 语言中使用异常的大多数建议都认为，在确信异常可以被捕获的任何情况下，应该优先使用检查型异常。语言设计（编译器强制您在方法签名中列出可能被抛出的所有检查型异常）以及早期关于样式和用法的著作都支持该建议。最近，几位著名的作者已经开始认为非检查型异常在优秀的 Java 类设计中有着比以前所认为的更为重要的地位。在本文中，Brian Goetz 考察了关于使用非检查型异常的优缺点。与 C 类似，Java 语言也提供异常的抛出和捕获。但是，与 C 不一样的是，Java 语言支持检查型和非检查型异常。Java 类必须在方法签名中声明它们所抛出的任何检查型异常，并且对于任何方法，如果它调用的方法抛出一个类型为 E 的检查型异常，那么它必须捕获 E 或者也声明为抛出 E（或者 E 的一个父类）。通过这种方式，该语言强制我们文档化控制可能退出一个方法的所有预期方式。对于因为编程错误而导致的异常，或者是不能期望程序捕获的异常（解除引用一个空指针，数组越界，除零，等等），为了使开发人员免于处理这些异常，一些异常被命名为非检查型异常（即那些继承自 RuntimeException 的异常）并且不需要进行声明。传统的观点在下面的来自 Sun 的“ The Java Tutorial ”的摘录中，总结了关于将一个异常声明为检查型还是非检查型的传统观点（更多的信息请参阅 参考资料）：因为 Java 语言并不要求方法捕获或者指定运行时异常，因此编写只抛出运行时异常的代码或者使得他们的所有异常子

类都继承自 RuntimeException，对于程序员来说是有吸引力的。这些编程捷径都允许程序员编写 Java 代码而不会受到来自编译器的所有挑剔性错误的干扰，并且不用去指定或者捕获任何异常。尽管对于程序员来说这似乎比较方便，但是它回避了 Java 的捕获或者指定要求的意图，并且对于那些使用您提供的类的程序员可能会导致问题。 检查型异常代表关于一个合法指定的请求的操作的有用信息，调用者可能已经对该操作没有控制，并且调用者需要得到有关的通知 例如，文件系统已满，或者远端已经关闭连接，或者访问权限不允许该动作。 如果您仅仅是因为不想指定异常而抛出一个 RuntimeException，或者创建 RuntimeException 的一个子类，那么您换取到了什么呢？您只是获得了抛出一个异常而不用您指定这样做的能力。换句话说，这是一种用于避免文档化方法所能抛出的异常的方式。在什么时候这是有益的？也就是说，在什么时候避免注明一个方法的行为是有益的？答案是“ 几乎从不。 ”换句话说，Sun 告诉我们检查型异常应该是准则。该教程通过多种方式继续说明，通常应该抛出异常，而不是 RuntimeException 除非您是 JVM。在 Effective Java: Programming Language Guide 一书中，Josh Bloch 提供了下列关于检查型和非检查型异常的知识点，这些与“ The Java Tutorial ”中的建议相一致（但是并不完全严格一致）： 第 39 条：只为异常条件使用异常。也就是说，不要为控制流使用异常，比如，在调用 Iterator.next() 时而不是在第一次检查 Iterator.hasNext() 时捕获 NoSuchElementException。第 40 条：为可恢复的条件使用检查型异常，为编程错误使用运行时异常。这里，Bloch 回应传统的 Sun 观点 运行时异常应该只是

用于指示编程错误，例如违反前置条件。第 41 条：避免不必要的使用检查型异常。换句话说，对于调用者不可能从其中恢复的情形，或者惟一可以预见的响应将是程序退出，则不要使用检查型异常。第 43 条：抛出与抽象相适应的异常。换句话说，一个方法所抛出的异常应该在一个抽象层次上定义，该抽象层次与该方法做什么相一致，而不一定与方法的底层实现细节相一致。例如，一个从文件、数据库或者 JNDI 装载资源的方法在不能找到资源时，应该抛出某种 ResourceNotFound 异常（通常使用异常链来保存隐含的原因），而不是更底层的 IOException、SQLException 或者 NamingException。 