河南中医药大学智能医学工程专业《计算机程序设计》课程

第05章：程序调试

黄子杰

河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设

河南中医药大学信息技术学院互联网技术教学团队

https://webdev.hactcm.edu.cn

2024/9/18

本章概要

- 在Java中进行程序调试通常涉及以下几个步骤：

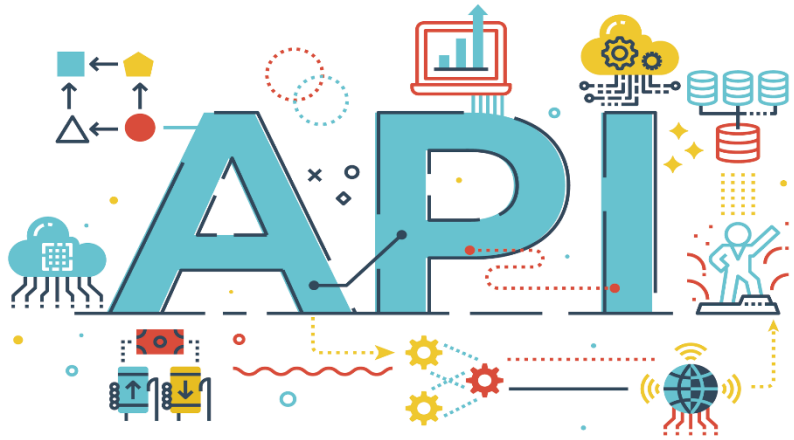
1 设置断点：在代码编辑器中，点击行号旁边的空白区域可以添加断点。

2 启动调试模式：在IDE中，通常可以通过点击调试图标或使用快捷键来启动调试会话。

3 逐行执行代码：使用调试工具栏上的按钮，如“步进”(Step Over)、“进入”(Step Into)、“跳出”(Step Out)，可以逐行执行代码。

4 观察变量：在调试期间，可以查看和修改变量的值。

5结束调试：当调试结束后，可以停止调试会话。



5.1 异常概述

- 概述

异常（Exception）是指程序在运行过程中发生的不正常情况，通常会导致程序的中断。Java使用异常处理机制来捕获和处理这些异常，以便程序能够继续运行或安全地终止。Java中的异常类主要分为两类：受检异常（Checked Exception）和非受检异常（Unchecked Exception）。

受检异常：在编译时检查的异常，通常是外部原因导致的，如文件未找到、网络连接失败等。必须使用try-catch处理或通过throws声明。

非受检异常：在运行时才会抛出的异常，通常是编程错误引起的，如NullPointerException、ArrayIndexOutOfBoundsException等。可选择处理或不处理。

5.1 异常概述

- 异常的生命周期与传播

异常的生命周期始于其被抛出的那一刻，直到被捕获或导致程序终止。当一个方法无法处理异常时，它可以将异常向上抛给调用者，这种传播机制确保了异常能够被适当处理，避免程序崩溃。

- Java中的异常处理机制

Java通过try-catch-finally语句来处理异常。try块中放置可能抛出异常的代码，catch块用于捕获并处理异常，finally块则无论是否发生异常都会执行，常用于资源的释放。此外，可以通过throw和throws关键字手动抛出或声明异常。

5.1 异常概述

- 异常处理的重要性

异常处理是确保程序健壮性的关键。通过合理地处理异常，程序可以在遇到错误时提供更多的信息，并且可以采取措施以防止程序完全崩溃。良好的异常处理策略可以提高程序的用户体验和可靠性。

- 异常处理的常见误区

在异常处理中，常见的误区包括不处理异常、捕获过于一般的异常、不正确地使用finally块等。这些误区可能会导致程序在遇到错误时无法正确响应，或者在某些情况下无法释放资源。

5.1 异常概述

- 尽早捕获异常

异常应该尽可能早地被捕获和处理。这有助于在问题发生时立即采取行动，而不是在程序的其他部分导致更多的错误。

- 精确捕获异常

应该只捕获那些需要处理的异常类型。捕获过于一般的异常（如捕获所有的Exception）可能会导致错误被错误地处理。

- 异常处理的文档化

在代码中，应该对异常处理进行充分的注释和文档化，以便其他开发者理解异常处理的逻辑。

- 资源管理

在异常处理中，应该确保所有使用的资源（如文件句柄、网络连接）都被正确地管理，并在不再需要时释放。

5.1 异常概述

- 生活举例：

- 异常概述：就像你在驾驶汽车时遇到突然爆胎，这就是一种“异常”情况。你需要采取行动（比如换胎）来继续行驶或安全停下。

5.1 异常概述

- 面试题示例：

问：什么是异常？Java中的异常如何分类？

答：异常是程序运行时出现的不正常情况。Java中的异常分为受检异常和非受检异常，受检异常必须处理或声明，非受检异常可以选择性处理。boolean: 占用 1 位，但在实际应用中会占用 1 字节（8 位），用于表示 true 或 false。它通常用于条件判断和逻辑操作。

5.2 使用try和catch捕获异常

- 概述：

try和catch是Java中用于捕获和处理异常的关键字。try块包含可能抛出异常的代码，而catch块用于捕获和处理该异常。

- 代码示例：

eg.5.2.txt

5.2 使用try和catch捕获异常

- try-catch语句的作用

try块用于封装可能抛出异常的代码，而catch块则用于捕获并处理这些异常。这种机制可以让程序在遇到异常时不会直接崩溃，而是可以优雅地处理异常情况，保证程序的健壮性。

- try-catch语句的使用场景

在实际编程中，当我们进行文件操作、网络请求等可能抛出异常的操作时，就需要使用try-catch语句来捕获可能发生的异常，并根据异常类型做出相应的处理。

5.2 使用try和catch捕获异常

- 生活举例：

- try-catch：就像你在驾驶时遇到突发情况（如突然刹车），你可以通过提前准备的应急措施（如控制车速）来避免事故。

5.2 使用try和catch捕获异常

- 面试题示例：

问：try和catch的作用是什么？

答：try块包含可能发生异常的代码，而catch块用于捕获和处理异常，防止程序崩溃。

5.3 使用throw和throws引发异常

- 概述

throw关键字用于显式抛出一个异常，通常在方法内部使用；而throws关键字用于在方法声明时指明该方法可能抛出的异常。

- 代码示例：

eg.5.3.txt

5.3 使用throw和throws引发异常

- 显式抛出异常

throw关键字用于在方法内部显式地抛出一个异常。当我们检测到某些错误情况时，可以使用throw来抛出一个异常，通知调用者当前方法遇到了问题。

- throw的使用场景

throw通常用于处理那些不应该在当前方法内部处理的错误情况。例如，当方法的输入参数不合法时，可以抛出一个IllegalArgumentException。

- throw的注意事项

使用throw抛出异常时，必须确保异常类型与catch块中声明的异常类型匹配，否则编译器将报错。

5.3 使用throw和throws引发异常

- 声明方法可能抛出的异常

throws关键字用于在方法声明中指明该方法可能抛出的异常类型。这样做可以让调用者知道该方法可能会抛出哪些异常，从而在调用时做好异常处理的准备。

- throws的注意事项

使用throws声明异常时，并不需要处理这些异常，但调用该方法时必须处理或再次声明这些异常。

- throws的使用场景

当一个方法可能抛出多个异常时，可以使用throws关键字来声明这些异常，而不是在每个方法内部单独处理它们。

5.3 使用throw和throws引发异常

- 生活举例:

- throw和throws：就像在机场安检时，如果发现违禁品（throw），安检员会发出警报（throws）来阻止携带者进入。

5.3 使用throw和throws引发异常

- 面试题示例：

问：throw和throws有什么区别？

答：throw用于显式地抛出异常，而throws用于声明一个方法可能抛出的异常。

5.4 finally关键字

- 概述

finally块是与try-catch结构一起使用的，它包含的代码在异常处理后总是执行，无论是否抛出异常。finally通常用于释放资源，如关闭文件或网络连接。

- 代码示例：

eg.5.4.txt

5.4 finally关键字

- 确保代码总是执行

finally块用于封装那些无论是否抛出异常都需要执行的代码。即使在try块或catch块中发生了异常，finally块中的代码也会被执行。

- finally块的执行时机

finally块在try块和catch块执行完毕后执行，但在返回语句执行之前。

5.4 finally关键字

- finally块的使用场景

finally块通常用于释放资源，如关闭文件或网络连接。即使在处理异常时，也需要确保这些资源被正确地释放。

- finally块的注意事项

如果finally块中包含return语句，它将覆盖try块或catch块中的return语句。因此，应谨慎在finally块中使用return。

5.4 finally关键字

- 生活举例：

- finally：就像在乘坐公交车时，不论途中发生什么情况（如堵车或事故），你在到站后都会下车（finally），这是一个必然的结果。

5.4 finally关键字

- 面试题示例：

问：finally块中的代码是否一定会执行？是否有例外情况？

答：finally块中的代码通常会执行，但如果JVM退出或在finally块之前发生系统级别的错误，则可能不执行。

5.5 getMessage和printStackTrace方法

- 概述：

getMessage()方法用于获取异常的简要描述信息，而printStackTrace()方法用于打印异常的堆栈跟踪信息，这有助于调试程序。

- 代码示例：

eg.5.5.txt

5.5 getMessage和printStackTrace方法

- 获取异常的简要描述

getMessage()方法用于获取异常的简要描述信息。它返回一个字符串，其中包含了异常的基本信息。

- getMessage的使用场景

在异常处理中，可以使用getMessage()方法来向用户显示一个简短的错误消息，或者用于日志记录。

- getMessage的注意事项

getMessage()方法返回的字符串应该足够简洁，以便用户能够快速理解错误的原因。

5.5 getMessage和printStackTrace方法

- 打印异常的堆栈跟踪信息

printStackTrace()方法用于打印异常的堆栈跟踪信息，这有助于调试程序。它会显示异常发生时调用栈的完整信息。

- printStackTrace的使用场景

在开发过程中，可以使用printStackTrace()方法来查看异常的详细堆栈信息，以便定位错误。

- printStackTrace的注意事项

printStackTrace()方法通常不用于生产环境，因为它可能会泄露敏感的内部信息。

5.5 getMessage和printStackTrace方法

- 生活举例：

- getMessage和printStackTrace：就像在手机上收到错误提示时，getMessage是简短的错误描述，而printStackTrace则是详细的技术报告。String 是 Java 的内置类，用于存储字符序列。

5.5 getMessage和printStackTrace方法

- 面试题示例：

问：getMessage()和printStackTrace()有什么区别？length(): 返回字符串的长度。

答：getMessage()返回异常的简要描述，而printStackTrace()则输出异常的详细堆栈信息，帮助定位错误。toUpperCase(): 将字符串转换为大写。

5.6 多重catch

- 概述

在try块后可以跟随多个catch块，以捕获不同类型的异常。每个catch块处理一种特定的异常类型。

- 代码示例：

eg.5.6.txt

5.6 多重catch

处理多个异常类型

- 多重catch的语法
- 在Java 7及以上版本中，可以在try块后跟随多个catch块，每个catch块处理一种特定的异常类型。

- 多重catch的使用场景

当一个代码块可能抛出多种类型的异常时，可以使用多重catch来分别处理这些异常。

- 多重catch的注意事项

在使用多重catch时，应该按照异常的继承层次结构从最具体的异常类型到最一般的异常类型排序。

5.6 多重catch

- 优化异常处理逻辑

通过使用多重catch，可以优化异常处理逻辑，避免在单个catch块中处理多种异常类型。

- 提高代码的可读性

多重catch可以提高代码的可读性，因为它允许开发者清晰地看到每种异常类型是如何被处理的。

- 多重catch的限制

多重catch不能处理同一个异常类型的多个实例，每个异常类型只能有一个catch块。

5.6 多重catch

- 生活举例：

- 多重catch：就像在面对不同的故障类型（如车辆爆胎或没油），你会采取不同的应对措施（如换胎或加油）。

5.6 多重catch

- 面试题示例：

问：什么是多重catch？如何使用？

答：多重catch用于处理多个异常类型，try块后可以跟随多个catch块，每个块处理一种特定的异常。

5.7 自定义异常类

- 概述

Java允许开发者创建自定义异常类，以满足特定的异常处理需求。自定义异常类通常继承自Exception或RuntimeException类。

- 代码示例：

eg.5.7.txt

5.7 自定义异常类

- 继承Exception或RuntimeException

创建自定义异常类时，通常需要继承自Exception类或其子类，或者继承自RuntimeException类。

- 自定义异常类的构造方法

自定义异常类应该提供至少一个构造方法，以便在抛出异常时可以传递异常信息。

- 自定义异常类的使用场景

自定义异常类用于处理特定的业务逻辑异常，使代码更具可读性和维护性。

5.7 自定义异常类

- 提高异常处理的精确度

自定义异常类可以提高异常处理的精确度，因为它允许开发者针对特定的业务逻辑创建异常。

- 增强代码的可维护性

通过使用自定义异常类，可以增强代码的可维护性，因为它们提供了更多的上下文信息。

- 自定义异常类的注意事项

自定义异常类应该遵循Java异常处理的最佳实践，包括提供清晰的错误信息和合理的异常继承结构。

5.7 自定义异常类

- 生活举例：

- 自定义异常类：就像你在自己家里制定的规则，如果某人违反了这些规则，你可以自定义处罚措施（自定义异常）。

5.7 自定义异常类

- 面试题示例：

问：如何创建自定义异常类？为什么需要自定义异常？

答：创建自定义异常类时，继承Exception或RuntimeException类并实现构造方法。自定义异常可以处理特定的业务逻辑异常，使代码更具可读性和维护性。这样详细讲解程序调试相关内容，可以帮助学生从基础到深入掌握异常处理的各个方面，通过实际代码示例和生活中的类比来增强理解。

5.7 自定义异常类

- 本章作业

作业一：异常捕获与处理实践

要求：

编写一个Java程序，该程序包含至少两个可能抛出异常的方法。

在主程序中调用这些方法，并使用try-catch语句捕获并处理这些异常。

至少包含一个finally块，用于执行一些必要的清理工作，比如关闭资源。

使用getMessage()和printStackTrace()方法打印异常信息，以便更好地了解异常发生的原因。

示例场景：

第一个方法尝试读取一个不存在的文件，可能抛出FileNotFoundException。

第二个方法尝试将字符串转换为整数，但字符串实际上不是一个有效的整数，可能抛出NumberFormatException。

5.7 自定义异常类

- 本章作业

作业二：自定义异常类与多重catch实践

要求：

定义一个自定义异常类MyCustomException，继承自Exception类。

编写一个Java程序，该程序包含多个可能抛出不同类型异常的方法，包括Java标准异常和你在第一步中定义的自定义异常。

使用多重catch块来捕获并处理这些不同类型的异常。

在每个catch块中，使用适当的消息来区分不同的异常类型，并给出处理建议。

示例场景：

方法一可能抛出自定义异常MyCustomException，表示某种特定于应用的错误情况。

方法二可能抛出IOException，比如在进行文件操作时遇到错误。

方法三可能抛出NullPointerException，比如尝试访问未初始化的对象

计算机程序设计课程学习平台

面向河南中医药大学智能医学工程专业使用

河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设

河南中医药大学信息技术学院互联网技术教学团队