

第九章 异常

1 异常

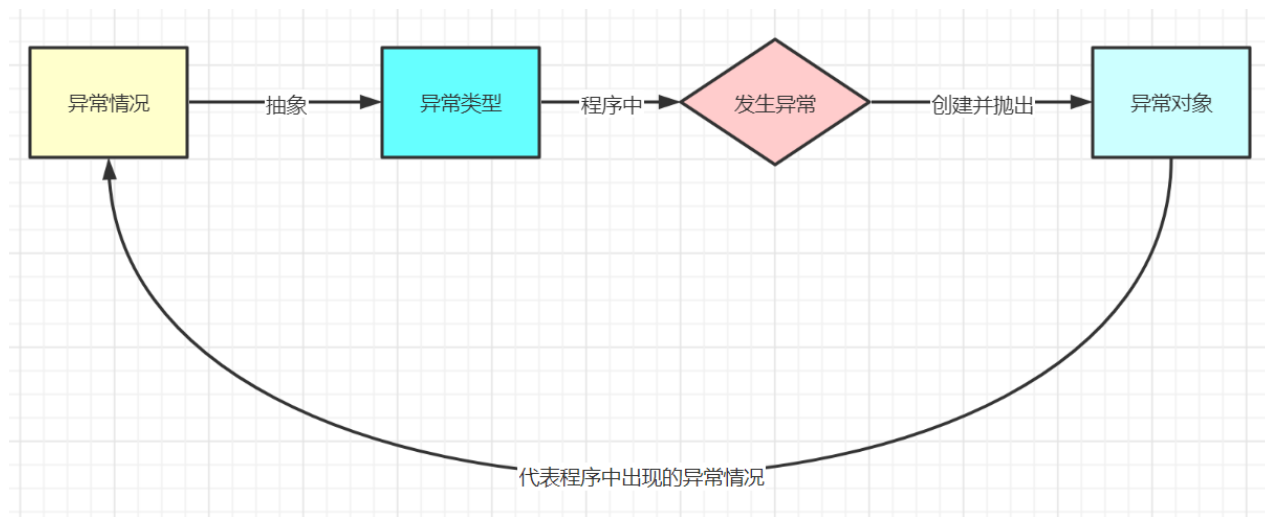
1.1 概述

程序在运行过程中，由于意外情况导致程序发生异常事件，默认情况下发生的异常会中断程序的运行。

在Java中，把常见的异常情况，都抽象成了对应的异常类型，那么每种异常类型都代表了一种特定的异常情况。

当程序中出现一种异常情况时，也会创建并抛出一个异常类型对象，这个对象就表示当前程序所出现的问题。

如图：



例如，程序中有一种异常情况是，当前使用下标从数组中取值的时候，这个下标值超过了数组下标的最大值，那么程序中就出现了异常情况，java中把这种异常情况抽象成了一个类：`java.lang.ArrayIndexOutOfBoundsException`，将来这个类的对象，就表示程序中出现了数组下标超过边界的异常情况。

案例描述：

观察下面各种异常情况。

```
1  //如何理解异常：
2  // 程序不正常情况，统称为 异常
3  public class Test011_Basic {
4      public static void main(String[] args) {
5          // ArithmeticException
6          int a = 10 / 0;
7
8          String s = "123";
9          //: NumberFormatException
10         int n = Integer.parseInt(s);
11
12         Object obj = new Object();//new String("hello");
13         //类型转换异常: ClassCastException
14         s = (String)obj;
15
16         int[] arr = {1,2,3,4};
17
18         arr = null;
19         //空指针异常: NullPointerException
20         System.out.println(arr[0]);
21
22         //数组索引越界 ArrayIndexOutOfBoundsException
23         System.out.println(arr[4]);
24     }
25 }
26
27 //运行结果：
28 Exception in thread "main" java.lang.ArithmeticException: / by
    zero
29     at
    com.briup.chap09.Test011_Basic.main(Test01_Basic.java:9)
```

可以看出，当前程序出现异常情况时，会创建并抛出和该异常情况对应的异常类的对象，这个异常对象中保存了一些信息，用来表示当前程序到底发生了什么异常情况。

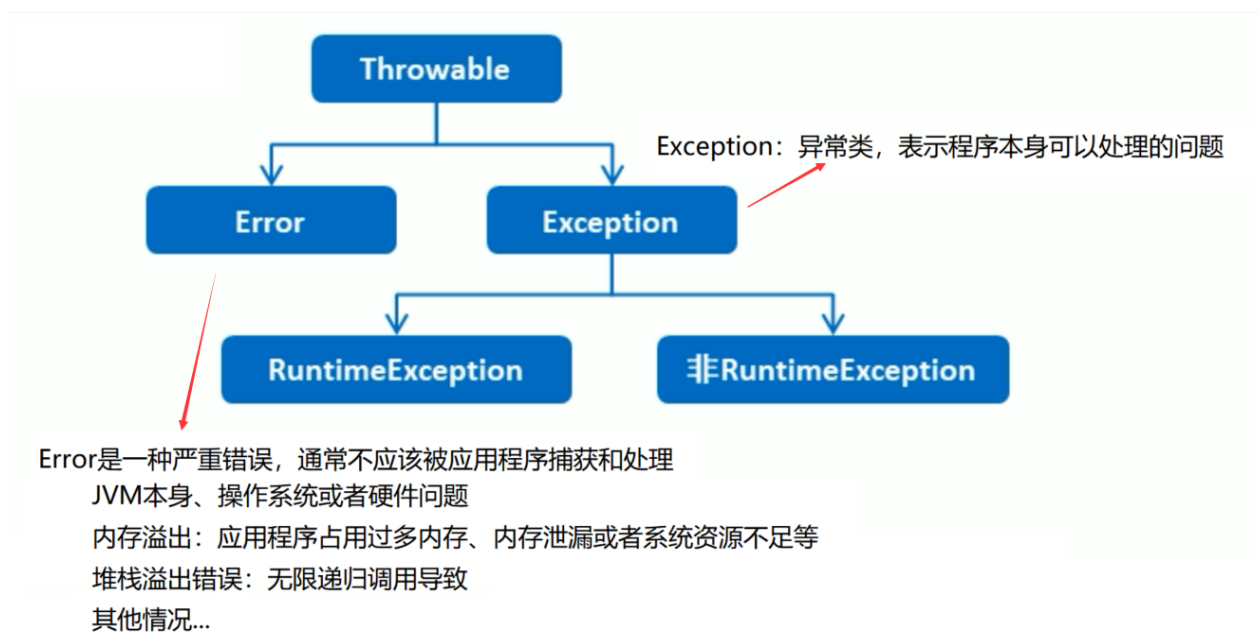
通过异常信息，我们可以定位异常发生的位置，以及异常发生的原因

1.2 异常体系

异常体系中的根类是： `java.lang.Throwable`，该类下面有两个子类型，`java.lang.Error` 和 `java.lang.Exception`

注意， `Throwable` 表示可以被抛出的

- `Error`，表示错误情况，一般是程序中出现了比较严重的问题，并且程序自身并无法进行处理。
- `Exception`，表示异常情况，程序中出了这种异常，大多是可以通过特定的方式进行处理和纠正的，并且处理完了之后，程序还可以继续往下正常运行。



注意，我们一般说的异常，都是指的Exception

1.3 异常种类

我们平时使用的异常类型，都是 `Exception` 类的子类型，它们把异常划分成了两种：

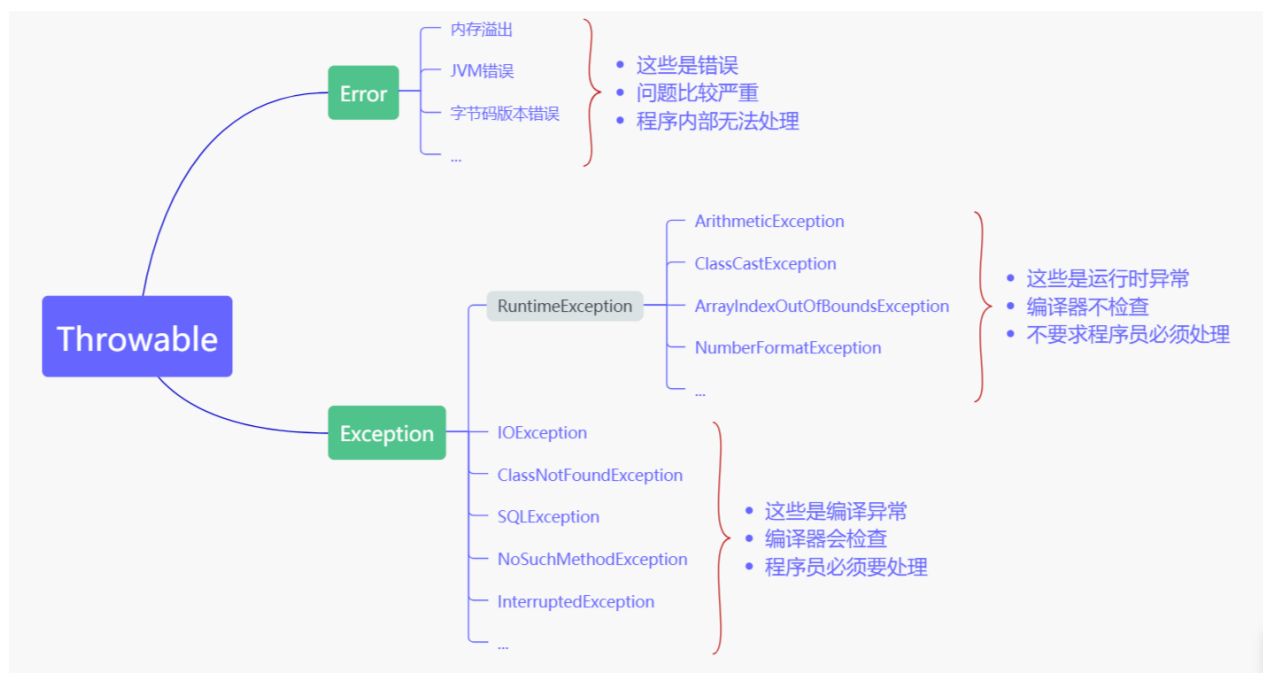
- 编译时异常
- 运行时异常

编译时异常

- 继承自 `Exception` 类，也称为checked exception
- 编译器在编译期间，会主动检查这种异常，如果发现异常则必须显示处理，否则程序就会发生错误，无法通过编译

运行时异常

- `RuntimeException` 类及其子类，也称为unchecked exception
- 编译器在编译期间，不会检查这种异常，也不要求我们去处理，但是在运行期间，如果出现这种异常则自动抛出



1.4 异常传播

如果一个方法中出现了异常的情况，系统默认的处理方式是：自动创建异常对象，并将这个异常对象抛给当前方法的调用者，并一直向上抛出，最终传递给JVM，JVM默认处理步骤有2步：

- 把异常的名称，错误原因及异常出现的位置等信息输出在了控制台
- 程序停止执行

案例展示：

```
1  package com.briup.chap09;
2
3  public class Test02_Default {
4      public static void main(String[] args) {
5          System.out.println("hello");
6          test1();
7          System.out.println("world");
8      }
9
10     public static void test1(){
11         test2();
12     }
13
14     public static void test2(){
15         test3();
16     }
17
18     public static void test3(){
19         //下面代码会抛出异常
20         int a = 1/0;
21     }
22 }
```

运行效果：

```
<terminated> Test02_Default [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月11日 下午6:48:35 - 下午6:48:35)
hello
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.briup.chap09.Test02_Default.test3(Test02_Default.java:20)
    at com.briup.chap09.Test02_Default.test2(Test02_Default.java:15)
    at com.briup.chap09.Test02_Default.test1(Test02_Default.java:11)
    at com.briup.chap09.Test02_Default.main(Test02_Default.java:6)
```

代码执行步骤解析：

- 因为 `java.lang.ArithmeticException` 是运行时异常，所以代码可以编译通过
- 程序运行时，先输出"hello"，然后一层一层调用，最终执行test3方法
- 执行test3方法时，出现除数为0的情况，系统自动抛出异常
`java.lang.ArithmeticException`
- 代码中没有对异常进行任何捕获处理，所以该异常往上传递给test2 --> test1 -> main --> JVM
- JVM虚拟机拿到异常后，输出异常相关信息，然后终止程序

2 异常抛出

2.1 自动抛出

Java代码中，出现了提前指定好的异常情况的时候，代码会自动创建异常对象，并且将该异常对象抛出。

例如，上述案例中执行 `int a = 1/0;` 的时候，代码会自动创建并抛出 `ArithmeticException` 类型的异常对象，来表示当前的这种异常情况。（算术异常）

又如，代码中执行 `String str = null; str.toString();` 的时候，代码会自动创建并抛出 `NullPointerException` 类型的异常对象，来表示当前这种异常情况。（空指针异常）

2.2 手动抛出

以上描述的异常情况，都是JVM中提前规定好的，我们不需要干预，JVM内部自己就会创建并抛出异常对象。

但是在其他的一些情况下，我们也可以手动的创建并抛出异常对象，抛出后系统也会按照默认的方式去处理。

手动抛出异常固定格式：

```
throw 异常对象;
```

案例展示：

从键盘录入用户名和密码，如果不是"root"和"briup"，则主动抛出异常 `RuntimeException`。

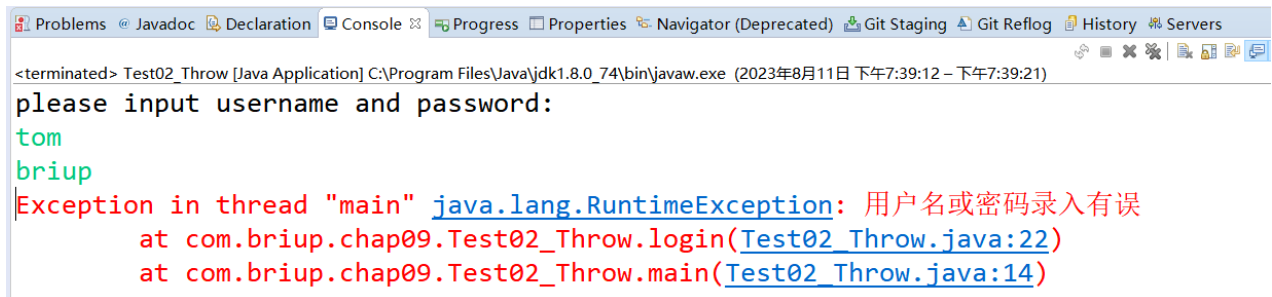
```
1 package com.briup.chap09;
2
3 import java.util.Scanner;
4
5 //手动抛出异常对象
6 public class Test02_Throw {
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         System.out.println("please input username and
10         password: ");
```

```

11         String username = sc.nextLine();
12         String password = sc.nextLine();
13
14         login(username,password);
15     }
16
17     public static void login(String name, String passwd) {
18         if("root".equals(name) && "briup".equals(passwd)) {
19             System.out.println("登录成功!");
20         }else {
21             //抛出运行时异常
22             throw new RuntimeException("用户名或密码录入有误");
23         }
24     }
25 }

```

运行效果：



```

<terminated> Test02_Throw [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月11日 下午7:39:12 - 下午7:39:21)
please input username and password:
tom
briup
Exception in thread "main" java.lang.RuntimeException: 用户名或密码录入有误
    at com.briup.chap09.Test02_Throw.login(Test02_Throw.java:22)
    at com.briup.chap09.Test02_Throw.main(Test02_Throw.java:14)

```

注意，此时方法中抛出的是一个运行时异常，编译器不会做出检查，所以代码可以正常的编译运行，但是运行的时候，如果用户名密码匹配失败，则抛出异常。

如果抛出的是编译时异常，则编译器会检查，代码无法通过编译，具体如下：


```

5 //手动抛出异常对象
6 public class Test02_Throw {
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         System.out.println("please input username and password: ");
10
11         String username = sc.nextLine();
12         String password = sc.nextLine();
13
14         login(username,password);
15     }
16
17     public static void login(String name, String passwd) {
18         if("root".equals(name) && "briup".equals(passwd)) {
19             System.out.println("登录成功!");
20         }else {
21             //抛出运行时异常
22             //throw new RuntimeException("用户名或密码录入有误");
23
24             //抛出编译时异常
25             throw new Exception("用户名或密码录入有误");
26         }
27     }
28 }

```

编译报错

如果要解决上述编译问题，需要程序员修改代码，有2种方式解决：

- 声明当前方法不对该异常处理，继续抛出异常，给上一级处理
- 主动捕获异常并处理

具体内容下一章节讲解。

3 异常处理

代码中出现了异常，除了默认的处理方式外，我们还可以手动处理异常：

- 声明继续抛出异常，借助throws关键字实现
- 捕获并处理异常，借助try、catch、finally关键字实现

3.1 throws

throws关键字用于在方法声明中指定该方法可能抛出的异常类型。

这个声明的目的，就是告诉方法的调用者，你调用我的这个方法的时候要小心啦，方法在运行的时候**可能会**抛出指定类型的异常。

定义格式：

```
1  [修饰符] 返回值类型 方法名(形参列表) throws 异常类名1,异常类名2... {
2      方法体语句;
3  }
```

案例展示：

使用throws关键字解决上述案例中编译报错的问题。

```
1 package com.briup.chap09;
2
3 import java.util.Scanner;
4
5 public class Test03_Login {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8         System.out.println("please input username and password: ");
9
10        String username = sc.nextLine();
11        String password = sc.nextLine();
12
13        login(username,password);
14    }
15
16    //当前方法中 存在编译时异常，如果程序员不处理无法通过编译
17    //现在程序员 借助throws关键字 声明 继续往上一级抛出异常，故而login方法编译通过
18    public static void login(String name, String passwd) throws Exception {
19        if("root".equals(name) && "briup".equals(passwd)) {
20            System.out.println("登录成功!");
21        } else {
22            //抛出Exception编译时异常
23            throw new Exception("用户名或密码录入有误");
24        }
25    }
26 }
```

思考：为什么此处编译报错

编译通过

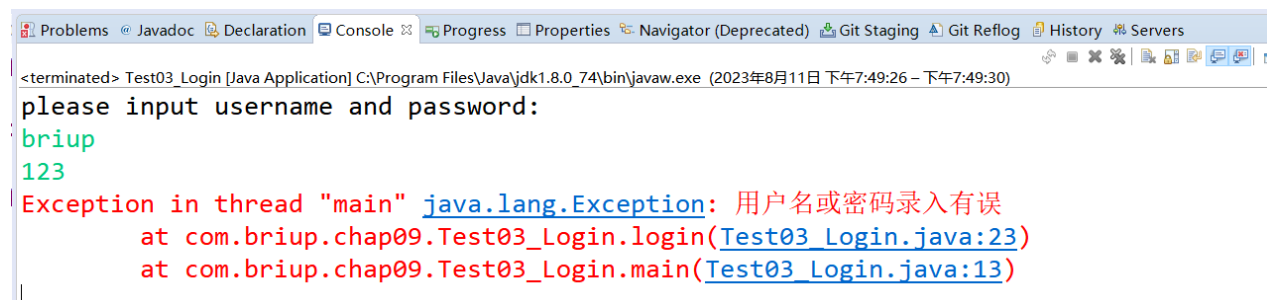
分析上述代码可知，login方法中虽然存在编译时异常，但程序员借助throws关键字告诉编译器，login方法中如果出现Exception异常则主动抛出，传递给上一级，所以login方法可以通过编译。

main方法中第13行之所以编译出错，是因为第13行可能会出现编译时异常，程序员必须主动处理该异常才可以：要么继续throws抛出，要么主动捕获处理（后续课程讲解）。

main方法借助throws声明抛出Exception解决异常：

```
5 public class Test03_Login {
6     public static void main(String[] args) throws Exception {
7         Scanner sc = new Scanner(System.in);
8         System.out.println("please input username and password: ");
9
10        String username = sc.nextLine();
11        String password = sc.nextLine();
12
13        login(username, password);
14    }
15
16    //当前方法中 存在编译时异常，如果程序员不处理无法通过编译
17    //现在程序员 借助throws关键字 声明 继续往上一次抛出异常，故而login方法编译通过
18    public static void login(String name, String passwd) throws Exception {
19        if("root".equals(name) && "briup".equals(passwd)) {
20            System.out.println("登录成功!");
21        } else {
22            //抛出Exception编译时异常
23            throw new Exception("用户名或密码录入有误");
24        }
25    }
26 }
```

运行效果：



```
<terminated> Test03_Login [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月11日 下午7:49:26 - 下午7:49:30)
please input username and password:
briup
123
Exception in thread "main" java.lang.Exception: 用户名或密码录入有误
    at com.briup.chap09.Test03_Login.login(Test03_Login.java:23)
    at com.briup.chap09.Test03_Login.main(Test03_Login.java:13)
```

常见面试题：throw和throws关键字区别

throw 关键字：

- `throw` 关键字用于手动抛出一个异常对象。它通常用在方法体内，用于在程序的任意位置抛出一个异常
- 使用 `throw` 关键字，可以将异常对象抛出到方法的调用者，然后由调用者处理该异常
- `throw` 关键字后面必须跟一个异常对象，该对象可以是Java内置的异常类，也可以是自定义的异常类的实例

throws 关键字：

- `throws` 关键字用于在方法签名中声明可能抛出的异常。它通常用于方法定义的位置，用于指示该方法有可能抛出指定的异常
- 使用 `throws` 关键字，可以将异常的处理责任交给方法的调用者，即告诉调用者该方法可能会抛出指定的异常，调用者需要进行相应的异常处理。
- `throws` 关键字后面跟的是一个异常类或多个异常类，使用逗号分隔

简而言之，`throw` 关键字用于手动抛出异常，而 `throws` 关键字用于声明方法可能抛出的异常。

`throw` 关键字将异常抛给方法的调用者，而 `throws` 关键字将异常的处理责任交给方法的调用者。

3.2 异常方法

`Exception` 中并没有定义方法，它的方法都是从 `Throwable` 中继承过来的，其中常用的方式有：

- `printStackTrace()`，打印输出当前发送异常的详细信息（重要）
- `getMessage()`，返回异常对象被抛出的时候，所携带的信息，一般是异常的发生原因（重要）

- `printStackTrace(PrintWriter s)`，方法重载，可以指定字符输出流，对异常信息进行输出
- `printStackTrace(PrintStream s)`，方法重载，可以指定字节输出流，对异常信息进行输出

下一个知识点，异常捕获中，我们就会用到这几个方法。

3.3 try-catch

`try-catch` 语句块，就是用来对指定代码，进行异常捕获处理，并且处理完成后，JVM不会停止运行，代码还可以正常的往下运行！

捕获异常语法：

```
1  try {  
2      可能会出现异常的代码；  
3  } catch (异常类型 引用名) {  
4      //处理异常的代码，可以是简单的输出异常信息  
5      //也可以使用日志进行了记录，也可以对数据进行修改纠正等操作  
6  
7      //一般输出异常信息  
8      //e.printStackTrace();  
9  }
```

try：该代码块中包含可能产生异常的代码

catch：用来进行某种类型异常的捕获，并对捕获到的异常进行处理

执行流程：

- 程序从 try 里面的代码开始执行

- 出现异常，就会跳转到对应的 `catch块` 里面去执行
- 执行完毕之后，程序出 `catch块`，继续往下执行

案例展示：

准备一个数组，从键盘录入1个索引值，输出数组在该位置上的值。

```
1  package com.briup.chap09;
2
3  import java.util.Scanner;
4
5  public class Test033_Catch {
6      public static void main(String[] args) {
7          Scanner sc = new Scanner(System.in);
8
9          int[] arr = {3,1,0,4,6,5};
10
11         try {
12             System.out.println("input index: ");
13             int index = sc.nextInt();
14
15             int num = arr[index];
16             System.out.println("arr["+index+"]: " + num);
17         } catch (IndexOutOfBoundsException e) {
18             System.out.println("in catch, catch异常成功...");
19             e.printStackTrace();
20             //测试异常类中其他几个方法
21             //System.out.println(e.getMessage());
22             //System.out.println(e.toString());
23         }
24
25         System.out.println("after try-catch ...");
26     }
27 }
```

运行正常效果：

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test033_Catch [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午3:32:56 – 下午3:33:00)
input index:
2
arr[2]: 0
after try-catch ...
```

异常效果：

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test033_Catch [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午3:36:34 – 下午3:36:36)
input index:
7
in catch, catch异常成功...
java.lang.ArrayIndexOutOfBoundsException: 7
after try-catch ...
    at com.briup.chap09.Test033_Catch.main(Test033_Catch.java:15)
```

观察上述输出可知，捕获异常成功，成功执行catch块中代码。

注意：try-catch语句处理过异常后，代码会继续往下执行，所以输出了"after try-catch ..."

上述案例中，键盘录入的如果不是整形数，而是字符串、字符或浮点数，运行效果如下：

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test033_Catch [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午3:39:16 – 下午3:39:24)
input index: 键盘录入一个字符串 或 字符 或 1.3
hello
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.briup.chap09.Test033_Catch.main(Test033_Catch.java:13)
```

思考：为什么没有捕获到对应的异常？

```

try {
    System.out.println("input index: ");
    int index = sc.nextInt();

    int num = arr[index];
    System.out.println("arr["+index+"]: " + num);
} catch (IndexOutOfBoundsException e) {
    System.out.println("in catch, catch异常成功...");
    e.printStackTrace();
}

```

只声明了 数组越界异常，则只能捕获这种异常
如果try块抛出其他异常，则无法捕获，继续往上抛出，JVM会用默认方式去处理

3.4 捕获多种异常

如果try语句块中的代码可能抛出多种异常，并且是不同类型的，则可以写多个catch语句块，用来同时捕获多种类型异常。

格式1:

```

1  try {
2      可能会出现异常的代码;
3  } catch (异常类型1 引用名) {
4      //处理异常的代码，可以是简单的输出异常信息
5  } catch (异常类型2 引用名) {
6      //异常处理代码
7  } ... {
8      //异常处理代码
9  } catch (异常类型n 引用名) {
10     //异常处理代码
11 }

```

案例展示:

上述案例优化，使可以捕获 `java.util.InputMismatchException` 异常。

```

1  package com.briup.chap09;
2
3  import java.util.InputMismatchException;

```

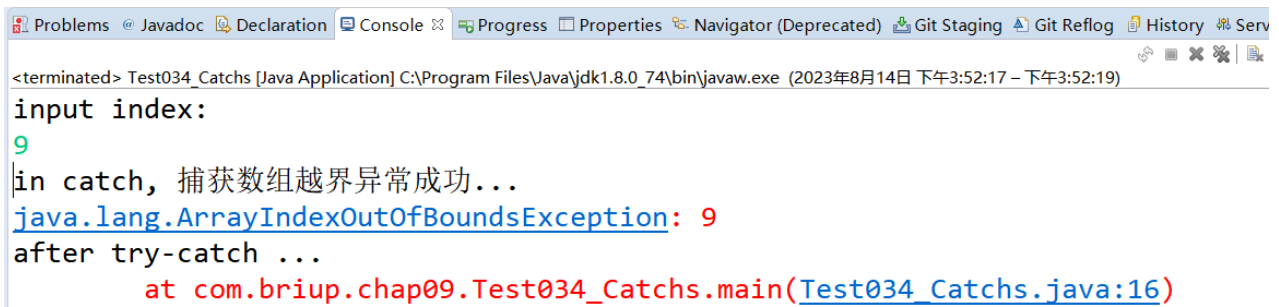


```

4  import java.util.Scanner;
5
6  public class Test034_Catches {
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9
10         int[] arr = {3,1,0,4,6,5};
11
12         try {
13             System.out.println("input index: ");
14             int index = sc.nextInt();
15
16             int num = arr[index];
17             System.out.println("arr["+index+"]: " + num);
18         } catch (IndexOutOfBoundsException e) {
19             System.out.println("in catch, 捕获数组越界异常成
功...");
20             e.printStackTrace();
21         } catch (InputMismatchException e) {
22             System.out.println("in catch, 捕获键盘录入格式有误异
常成功...");
23             e.printStackTrace();
24         }
25
26         System.out.println("after try-catch ...");
27     }
28 }

```

运行效果1：成功捕获数组越界异常



```

<terminated> Test034_Catches [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午3:52:17 - 下午3:52:19)
input index:
9
in catch, 捕获数组越界异常成功...
java.lang.ArrayIndexOutOfBoundsException: 9
after try-catch ...
    at com.briup.chap09.Test034_Catches.main(Test034_Catches.java:16)

```

运行效果2：成功捕获输入不匹配异常

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test034_Catches [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午3:55:56 - 下午3:56:00)
input index:
a
in catch, 捕获键盘录入格式有误异常成功...
java.util.InputMismatchException
after try-catch ...
    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.briup.chap09.Test034_Catches.main(Test034_Catches.java:14)
```

注意事项：

这种异常处理方式，要求多个catch中的异常不能相同

如果catch中的多个异常类之间有**子类父类**关系的话，那么子类异常必须写在父类异常**上面**的catch块中，父类异常必须写在**下面**的catch块中。

因为如果父类型异常再最上面的话，下面catch语句代码，永远不会被执行。

```
try {
    System.out.println("input index: ");
    int index = sc.nextInt();

    int num = arr[index];
    System.out.println("arr["+index+"]: " + num);
} catch (Exception e) { // 父类异常
    System.out.println("in catch, catch异常成功...");
    e.printStackTrace();
} catch (IndexOutOfBoundsException e) { // 子类异常
    System.out.println("in catch, catch异常成功...");
    e.printStackTrace();
}
```

编译报错

格式2：

相对格式1，书写更加紧凑

```

1  try {
2      可能会出现异常的代码;
3  }catch(异常类型1|异常类型2|...|异常类型n 引用名) {
4      //处理异常的代码, 可以是简单的输出异常信息
5  }

```

案例展示:

使用格式2对上述案例进行改造。

```

1  package com.briup.chap09;
2
3  import java.util.InputMismatchException;
4  import java.util.Scanner;
5
6  public class Test034_Catches {
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9
10         int[] arr = {3,1,0,4,6,5};
11
12         try {
13             System.out.println("input index: ");
14             int index = sc.nextInt();
15
16             int num = arr[index];
17             System.out.println("arr["+index+"]: " + num);
18         }catch(IndexOutOfBoundsException |
InputMismatchException e) {
19             if(e instanceof IndexOutOfBoundsException) {
20                 System.out.println("in catch, 捕获数组越界异常成
功...");
21             }else if(e instanceof InputMismatchException) {
22                 System.out.println("in catch, 捕获键盘录入格式有
误异常成功...");

```

```

23         }
24
25         e.printStackTrace();
26     }
27
28     System.out.println("after try-catch ...");
29 }
30 }

```

运行效果与之前一样，略...

结论：

实际开发中，异常的处理不会过于复杂，一般catch块中捕获Exception即可。因为Exception是最大的异常类型，由于多态的原因，Exception类型的引用e，可以捕获接收到任意类型的异常对象。

案例展示：

```

1  package com.briup.chap09;
2
3  public class Test034_Catches {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6          int[] arr = {3,1,0,4,6,5};
7
8          try {
9              System.out.println("input index: ");
10             int index = sc.nextInt();
11
12             int num = arr[index];
13             System.out.println("arr["+index+"]: " + num);
14         } catch (Exception e) {
15             e.printStackTrace();

```

```
16         }
17
18         System.out.println("after try-catch ...");
19     }
20 }
```

3.5 finally语句

finally 关键字可以和 **try**、**catch** 关键字一起使用，固定搭配为：**try-catch-finally**，它可以保证指定finally中的代码一定会执行，无论是否发生异常！

固定格式：

```
1  try {
2      可能出现异常的代码
3  }catch(异常类型 引用名) {
4      异常捕获后操作代码
5  }finally {
6      离开try或catch块前，必须要执行的代码
7  }
```

finally 块的主要作用：

- **资源释放**：在 **try** 块中打开的资源（例如文件、数据库连接、网络连接等）可以在 **finally** 块中关闭或释放，以确保资源的正确释放，即使在发生异常的情况下也能够执行释放操作。
- **清理操作**：**finally** 块可以用于执行一些清理操作，例如关闭打开的流、释放锁、取消注册监听器等。

- **异常处理的补充：** `finally` 块可以用于在 `try` 块和 `catch` 块之后执行一些必要的操作，例如记录日志、发送通知等。

案例展示：

运行下面程序，观察产生异常时、不产生异常时，`finally`代码块执行效果。

```
1  package com.briup.chap09;
2
3  public class Test035_Finally {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6
7          try {
8              System.out.println("请录入两个整数：");
9              int a = sc.nextInt();
10             int b = sc.nextInt();
11             int n = a / b;
12             System.out.println("n: " + n);
13         } catch (Exception e) {
14             e.printStackTrace();
15             System.out.println("即将离开catch块 ...");
16         } finally {
17             //无论try块中是否发生异常，finally块中的代码都会被执行
18             System.out.println("in finally 代码块 ...");
19         }
20
21         System.out.println("after try-catch-finally ...");
22     }
23 }
```

正常运行：

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test03_Finally [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午2:55:19 - 下午2:55:21)
请输入两个整数:
10 5
n: 2
in finally 代码块 ...
after try-catch-finally ...
```

finally代码块内容执行

异常执行:

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test03_Finally [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午2:56:05 - 下午2:56:07)
请输入两个整数:
2 0
java.lang.ArithmeticException: / by zero
即将离开catch块 ...
in finally 代码块 ...
after try-catch-finally ...
    at com.briup.chap09.Test03_Finally.main(Test03_Finally.java:13)
```

产生异常 finally代码块仍旧执行

常见面试题:

观察下面代码, 说明第7行输出的r为什么?

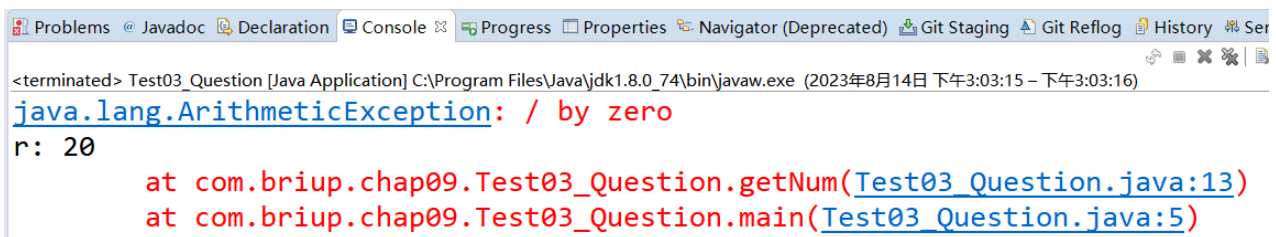
```
1 package com.briup.chap09;
2
3 public class Test036_Question {
4     public static void main(String[] args) {
5         int r = getNum(10,0);
6
7         System.out.println("r: " + r);
8     }
9
10    public static int getNum(int a, int b) {
11        int n = 0;
12        try {
13            n = a / b;
14        } catch (Exception e) {
15            e.printStackTrace();
16            n = 20;
```

```

17          //先建立返回通道，放入 n当前的值 20，在最终返回前，再去调
    用finally
18          return n;
19      }finally {
20          //System.out.println("in finally, n: " + n);
          //20
21          n = 30;
22          //System.out.println("in finally,最后 n: " + n);
          //30
23      }
24
25      return n;
26  }
27  }

```

运行结果：



```

<terminated> Test03_Question [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午3:03:15 - 下午3:03:16)
java.lang.ArithmeticException: / by zero
r: 20
    at com.briup.chap09.Test03_Question.getNum(Test03_Question.java:13)
    at com.briup.chap09.Test03_Question.main(Test03_Question.java:5)

```

4 自定义异常

4.1 应用场景

JavaAPI中已经存在的异常类，都是当年sun公司，提前定义好的，它们分别表示着某一种已知的异常情况。

但是，在我们开发的系统中，大多数业务功能里面总会出现一些新的异常情况，而这些异常情况，当年sun公司定义异常类型的时候，肯定是想不到的。

例如，在学生信息管理系统中，学生的年龄设置为负数、学生的成绩大于了100分，用户登录时候的密码不正确、用户访问某些接口时候的权限不足等情况，在系统中都是异常情况，而这些异常类型在JavaAPI中都是没有的。

所以，在实际开发中，我们会自定义一些异常的类型，用来表示上面描述的那些异常情况，这样做的好处就是，我们通过观察系统的运行日志，就可以很快的知道当前系统是发生了什么事情，才导致出了这些异常情况。

4.2 自定义异常

- 如果要自定义一个编译时异常类型，就自定义一个类，并继承 `Exception`
- 如果要自定义一个运行时异常类型，就自定义一个类，并继承 `RuntimeException`

定义步骤：

1. 定义异常类
2. 写继承关系
3. 提供空参构造
4. 提供带参构造

例如，自定义编译时异常类型，通过名字可知，这是在用户登录期间发生异常时，应该创建并抛出的异常类型

```

1  public class LoginExceptin extends Exception{
2      public LoginExceptin() {
3
4      }
5
6      public LoginExceptin(String message) {
7          super(message);
8      }
9  }

```

案例展示:

准备一个Student学生类，包含数据成员age，其取值范围为[4,79]，如果setAge时参数取值不在该范围中，则抛出自定义运行时异常类型 **AgeOutOfRangeException** 对象。

```

1  package com.briup.chap09;
2
3  class Student {
4      private String name;
5      private int age;
6
7      public String getName() {
8          return name;
9      }
10     public void setName(String name) {
11         this.name = name;
12     }
13     public int getAge() {
14         return age;
15     }
16     public void setAge(int age) {
17         //设置age取值范围[4,79]
18         if(age < 3 || age > 80) {

```

```

19         //System.out.println("年龄不在有效范围");
20         //程序员 主动抛出异常:
21         // throw 异常对象;
22         throw new AgeOutOfRangeException("年龄不在有效范围");
23         //return;
24     }
25
26     this.age = age;
27 }
28
29 @Override
30 public String toString() {
31     return "Student [name=" + name + ", age=" + age + "]";
32 }
33 }
34
35 //自定义异常, 属于运行时异常, 年龄不在有效范围异常
36 class AgeOutOfRangeException extends RuntimeException {
37     public AgeOutOfRangeException() {}
38
39     public AgeOutOfRangeException(String message) {
40         super(message);
41     }
42 }
43
44 public class Test04_AgeException {
45     public static void main(String[] args) {
46         Student s1 = new Student();
47         s1.setName("zs");
48         s1.setAge(21);
49         System.out.println(s1);
50
51         System.out.println("-----");
52
53         Student s2 = new Student();
54         try {

```

```

55         s2.setName("tom");
56         s2.setAge(221);
57     }catch(AgeOutOfRangeException e) {
58         e.printStackTrace();
59     }
60
61     System.out.println(s2);
62 }
63 }

```

运行效果：

```

<terminated> Test04_AgeException [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午2:35:27 - 下午2:35:27)
Student [name=zs, age=21]
-----
com.briup.chap09.AgeOutOfRangeException: 年龄不在有效范围
    at com.briup.chap09.Student.setAge(Test04_AgeException.java:44)
    at com.briup.chap09.Test04_AgeException.main(Test04_AgeException.java:15)
Student [name=tom, age=0]

```

5 断言 assert

断言（`assert`），是JDK1.4的时候，增加的一个关键字。用它可以在程序中，确认一些关键性条件必须是成立的，否则会抛出 `AssertionError` 类型的错误。（了解即可）

注意，断言（`assert`）并不是用来代替 `if` 判断的，而是确认系统中的一些**关键性条件是必须成立的**，所以 `assert` 和 `if` 并不冲突，并且还可以通过给JVM传参数，来控制断言（`assert`）是否生效。

断言（`assert`）的使用方式：

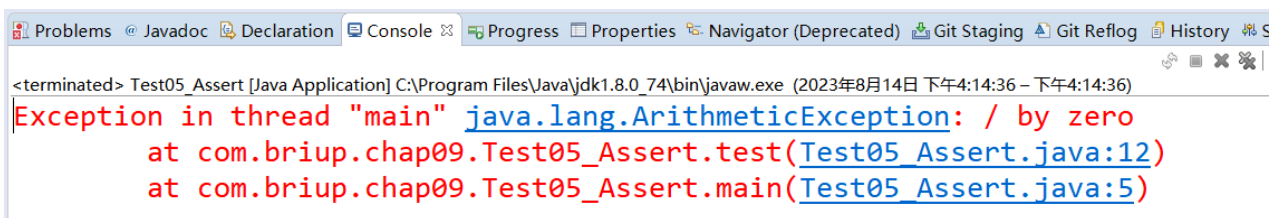
```
1  assert 布尔表达式;
2  //或者
3  assert 布尔表达式 : "错误信息";
```

当布尔表达式为true时，断言通过，否则抛出 `AssertionError` 类型错误
所以，assert后面的布尔表达式必须true才行。（也就是说条件必须成立）

案例展示:

```
1  package com.briup.chap09;
2
3  public class Test05_Assert {
4      public static void main(String[] args) {
5          test(0);
6      }
7
8      public static void test(int a) {
9          assert a!=0 : "参数a的值不能为0";
10
11         int b = 10;
12         int c = b/a;
13
14         System.out.println(c);
15     }
16 }
```

运行效果:



The screenshot shows the console output of a Java application. The title bar indicates the application is 'Test05_Assert [Java Application]' running on 'C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe' at '2023年8月14日 下午4:14:36 - 下午4:14:36'. The console text shows a terminated state followed by an exception: 'Exception in thread "main" java.lang.ArithmeticException: / by zero'. The stack trace points to 'at com.briup.chap09.Test05_Assert.test(Test05_Assert.java:12)' and 'at com.briup.chap09.Test05_Assert.main(Test05_Assert.java:5)'. The IDE interface includes tabs for Problems, Javadoc, Declaration, Console, Progress, Properties, Navigator (Deprecated), Git Staging, Git Reflog, History, and Source.

```
<terminated> Test05_Assert [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午4:14:36 - 下午4:14:36)
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.briup.chap09.Test05_Assert.test(Test05_Assert.java:12)
    at com.briup.chap09.Test05_Assert.main(Test05_Assert.java:5)
```

发现断言没有生效!

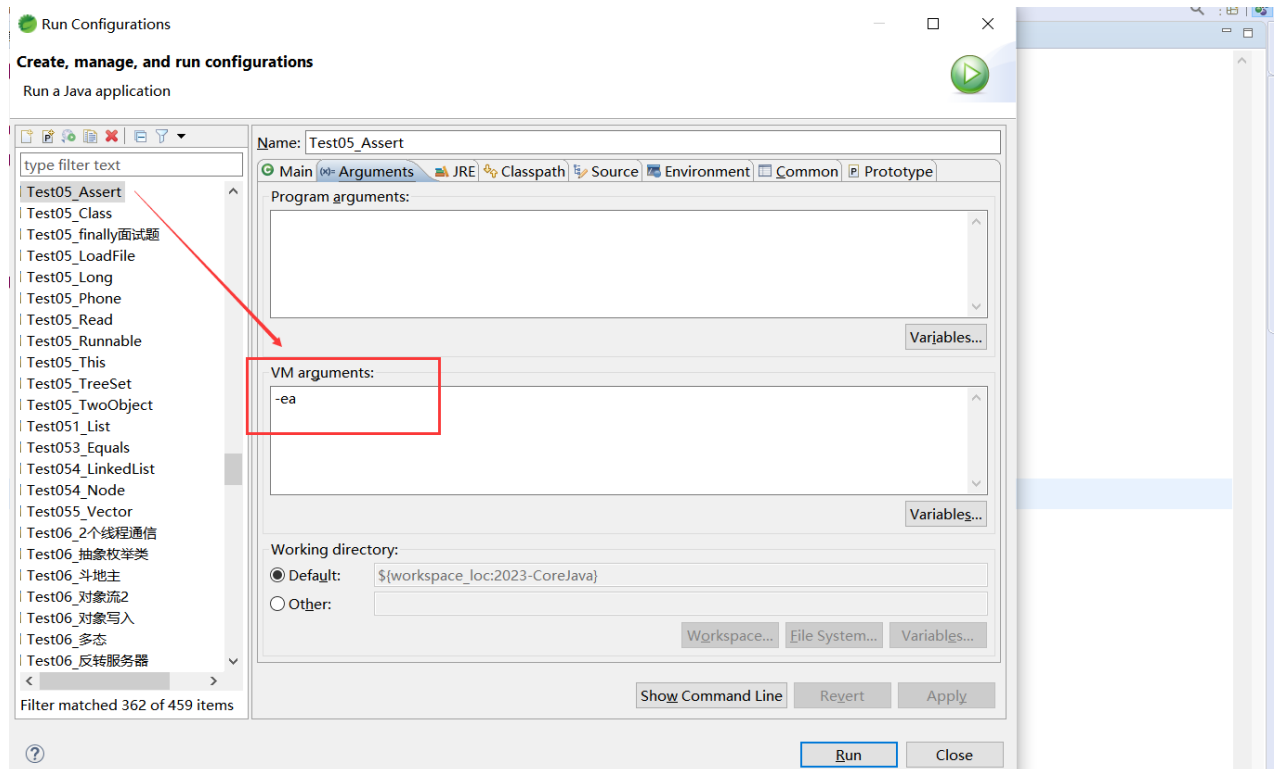
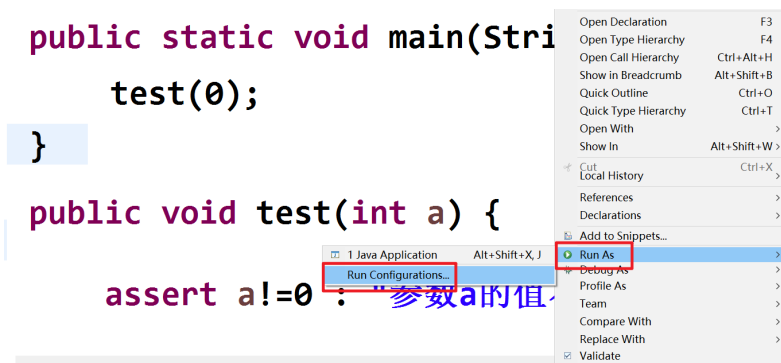
原因分析:

默认情况下, JVM没有开启断言功能, 需要通过给JVM传参打开此项功能

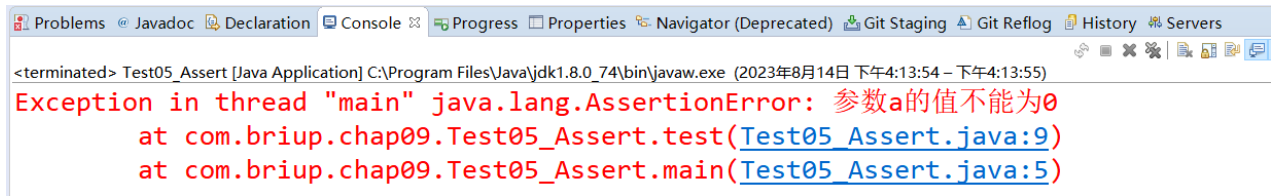
需要使用 `-enableassertions` 或者 `-ea` JVM参数

例如, `java -ea com.briup.demo.Test`

使用eclipse或sts运行时:



此时的运行结果为：因为断言要求参数a不能为0，但实际参数传的为0



The screenshot shows an IDE's console window with the following content:

```
<terminated> Test05_Assert [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年8月14日 下午4:13:54 - 下午4:13:55)  
Exception in thread "main" java.lang.AssertionError: 参数a的值不能为0  
    at com.briup.chap09.Test05_Assert.test(Test05_Assert.java:9)  
    at com.briup.chap09.Test05_Assert.main(Test05_Assert.java:5)
```

如果去掉-ea参数的话，那么断言（assert）语句，在JVM执行代码的时候，会被直接忽略的