异常（Exception）

@M了个J
李明杰

https://github.com/CoderMJLee
http://cnblogs.com/mjios

码拉松

实力IT教育 www.520it.com

小码哥教育@M了个J

开发中的错误

- 在开发 Java 程序的过程中，会遇到各种各样的错误
  - 语法错误
    - ✓ 会导致编译失败，程序无法正常运行
  - 逻辑错误
    - ✓ 比如需要执行加法操作时，不小心写成了减法操作
  - 运行时错误
    - ✓ 在程序运行过程中产生的意外，会导致程序终止运行
    - ✓ 在 Java 中也叫做异常

- 程序产生了异常，一般称之为：抛出了异常
  - 如果没有主动去处理它，会导致程序终止运行

```
public static void main(String[] args) {
    代码1
    代码2
    代码3
}
```

- 如果【代码2】抛出了异常，并且没有主动去处理它
  - 程序就会退出，【代码3】将没有机会执行

小码哥教育 SEEMYGO

思考：下面代码的打印结果是什么？

```
public static void main(String[] args) {
    System.out.println(1);
    Integer i1 = new Integer("123");
    System.out.println(2);
    Integer i2 = new Integer("abc");
    System.out.println(3);
}
```

- 由于 "abc" 无法转换成整数，new Integer("abc") 会抛出一个异常
  - 异常类型：java.lang.NumberFormatException
  - 由于没有主动去处理这个异常，所以导致程序终止运行
- ✓ 打印结果是：1、2

小码哥教育 @M了个M

小码哥教育 SEEMYGO

思考：下面代码的打印结果是什么？

```
public static void main(String[] args) {
    System.out.println(1);
    Integer i = new Integer("1234");
    Object obj = "12.34";
    Double d = (Double) obj;
    System.out.println(2);
}
```

- 由于 "12.34" 无法强转成 Double 类型，(Double) obj 会抛出一个异常
  - 异常类型：java.lang.ClassCastException
  - 由于没有主动去处理这个异常，所以导致程序终止运行
- ✓ 打印结果：1

小码哥教育 @M了个M

小码哥教育 SEEMYGO

思考：下面代码的打印结果是什么？

```
public static void main(String[] args) {
    Integer[] nums = { 11, null, 22 };
    for (int num : nums) {
        System.out.println(num);
    }
}
```

```
Integer[] nums = { 11, null, 22 };
// 建议的写法
for (Integer num : nums) {
    System.out.println(num);
}
// 打印结果：11、null、22
```

- Integer 在自动拆箱为 int 时，会调用 Integer 对象的 intValue() 方法
- 由于 nums[1] 为 null，使用 null 调用方法会抛出一个异常
  - 异常类型：java.lang.NullPointerException
  - 由于没有主动去处理这个异常，所以导致程序终止运行
- ✓ 打印结果：11

小码哥教育 SEEMYGO

打印的细节

```
public class Dog {
    @Override
    public String toString() {
        return "Dog - 666";
    }
}
```

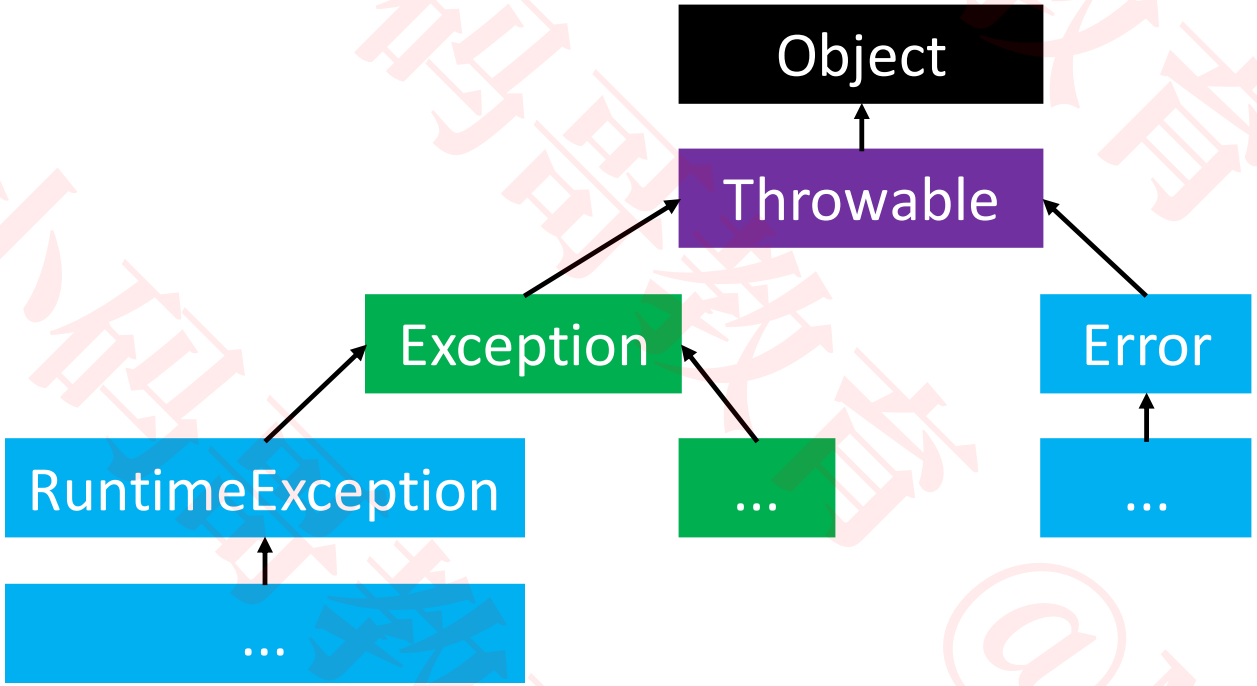
```
Dog dog1 = new Dog();
// Dog - 666
System.out.println(dog1);

Dog dog2 = null;
// null
System.out.println(dog2);
```

```
public void println(Object x) {
    String s = String.valueOf(x);
    synchronized (this) {
        print(s);
        newLine();
    }
}
```

```
public static String valueOf(Object obj) {
    return (obj == null) ? "null" : obj.toString();
}
```

异常（Exception）

- Java 中有各种各样的异常
  - 所有的异常最终都继承自 java.lang.Throwable



- 如何防止程序因为抛出异常导致终止运行?
  - 可以通过 `try-catch` 来捕捉处理异常

try-catch

```
try {
    代码1
    代码2（可能会抛出异常）
    代码3
} catch (异常A e) {
    // 当抛出【异常A】类型的异常时，会进入这个代码块

} catch (异常B e) {
    // 当没有抛出【异常A】类型
    // 但抛出【异常B】类型的异常时，会进入这个代码块

} catch (异常C e) {
    // 当没有抛出【异常A】、【异常B】类型
    // 但抛出【异常C】类型的异常时，会进入这个代码块
}
代码4
```

- 如果【代码2】没有抛出异常
  - ① 【代码1、3】都会被执行
  - ② 所有的 `catch` 都不会被执行
  - ③ 【代码4】会被执行

- 如果【代码2】抛出异常
  - ① 【代码1】会被执行、【代码3】不会被执行
  - ② 会选择匹配的 `catch` 来执行代码
  - ③ 【代码4】会被执行

- 父类型的异常必须写在子类型的后面
  - 【异常A】不可以是【异常B、C】的父类型
  - 【异常B】不可以是【异常C】的父类型

小码哥教育 SEEMYGO

思考：下面代码的打印结果是什么？

```
public static void main(String[] args) {
    System.out.println(1);
    try {
        System.out.println(2);
        Integer i = new Integer("abc");
        System.out.println(3);
    } catch (NumberFormatException e) {
        System.out.println(4);
    }
    System.out.println(5);
}
```

- 打印结果是：1、2、4、5

小码哥教育 SEEMYGO

一个 catch 捕获多种类型的异常

```
try {

} catch (异常A | 异常B | 异常C e) {
    // 当抛出【异常A】或【异常B】或【异常C】类型的异常时，会进入这个代码块

}
```

- 从 Java 7 开始，单个 `catch` 可以捕获多种类型的异常
- 如果并列的几个异常类型之间存在父子关系，保留父类型即可
- 这里的变量 e 是隐式 `final` 的

小码哥教育 SEEMYGO

异常对象的常用方法

```
try {
    Integer i = new Integer("abc");
} catch (NumberFormatException e) {
    // 异常描述
    System.out.println(e.getMessage());
    // 异常名称 + 异常描述
    System.out.println(e);
    // 打印堆栈信息
    e.printStackTrace();
}
```

finally

- try 或 catch 正常执行完毕后，一定会执行 finally 中的代码
- finally 可以和 try-catch 搭配使用，也可以只和 try 搭配使用
- 经常会在 finally 中编写一些关闭、释放资源的代码（比如关闭文件）

```
try {

} catch (异常 e) {

} finally {

}
```

```
try {

} finally {

}
```

```
PrintWriter out = null;
try {
    out = new PrintWriter("F:/mj.txt");
    out.print("My name is MJ.");
} catch (FileNotFoundException e) {
    e.printStackTrace();
} finally {
    if (out != null) {
        out.close();
    }
}
```

finally 细节

- 如果在执行 try 或 catch 时，JVM 退出或者当前线程被中断、杀死
  - finally 可能不会执行

- 如果 try 或 catch 中使用了 return、break、continue 等提前结束语句
  - finally 会在 return、break、continue 之前执行

小码哥教育 @M了个J

小码哥教育 SEEMYGO

思考：下面代码的打印结果是什么？

```
for (int i = 1; i <= 3; i++) {
    try {
        System.out.println(i + "_try_1");
        if (i == 2) continue;
        System.out.println(i + "_try_2");
    } finally {
        System.out.println(i + "_finally");
    }
}
```

- 打印结果是
  - 1_try_1
  - 1_try_2
  - 1_finally
  - 2_try_1
  - 2_finally
  - 3_try_1
  - 3_try_2
  - 3_finally

小码哥教育 SEEMYGO

思考：下面代码的打印结果是什么？

```
for (int i = 1; i <= 3; i++) {
    try {
        System.out.println(i + "_try_1");
        if (i == 2) break;
        System.out.println(i + "_try_2");
    } finally {
        System.out.println(i + "_finally");
    }
}
```

- 打印结果是
  - 1_try_1
  - 1_try_2
  - 1_finally
  - 2_try_1
  - 2_finally

小码哥教育 @M了个J

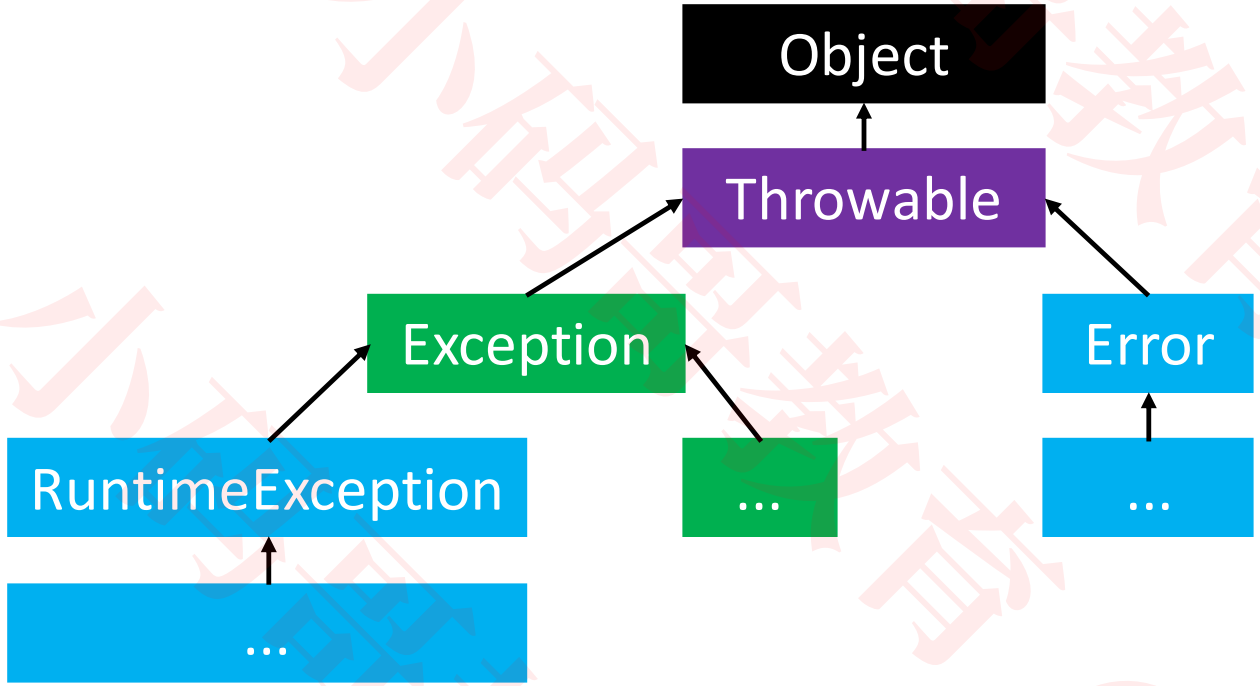
小码哥教育 SEEMYGO

思考：下面代码的打印结果是什么？

```
static int get() {
    try {
        new Integer("abc");
        System.out.println(1);
        return 2;
    } catch (Exception e) {
        System.out.println(3);
        return 4;
    } finally {
        System.out.println(5);
    }
}

public static void main(String[] args) {
    System.out.println(get());
}
```

- 打印结果是：3、5、4

异常的种类



- 检查型异常（Checked Exception）
  - 这类异常一般难以避免，编译器会进行检查
  - ✓ 如果开发者没有处理这类异常，编译器将会报错
  - 哪些异常是检查型异常?
  - ✓ 除 Error、RuntimeException 以外的异常
- 非检查型异常（Unchecked Exception）
  - 这类异常一般可以避免，编译器不会进行检查
  - ✓ 如果开发者没有处理这类异常，编译器将不会报错
  - 哪些异常是非检查型异常?
  - ✓ Error、RuntimeException

小码哥教育 SEEMYGO

常见的检查型异常

```
// java.io.FileNotFoundException，文件不存在
FileOutputStream fos = new FileOutputStream("F:/mj/520it.txt");
```

```
FileOutputStream fmt = new SimpleDateFormat("yyyy-MM-dd");
// java.text.ParseException，字符串的格式不对
Date date = fmt.parse("2066/06/06");
```

```
// java.lang.InterruptedException
Thread.sleep(1000);
```

```
// java.lang.ClassNotFoundException，不存在这个类
Class cls = Class.forName("Dog");
// java.lang.InstantiationException，没有无参构造方法
// java.lang.IllegalAccessException，没有权限访问构造方法
Dog dog = (Dog) cls.newInstance();
```

小码哥教育 SEEMYGO

常见的非检查型异常 – Error

```
for (int i = 0; i < 200; i++) {
    // java.lang.OutOfMemoryError，内存不够用
    long[] a = new long[1000000000];
}
```

```
public static void test() {
    test();
}

public static void main(String[] args) {
    // java.lang.StackOverflowError，栈内存溢出
    test();
}
```

小码哥教育 SEEMYGO

常见的非检查型异常 – RuntimeException

```
// java.lang.NullPointerException，使用了空指针
StringBuilder s = null;
s.append("abc");
```

```
// java.lang.NumberFormatException，数字的格式不对
Integer i = new Integer("abc");
```

```
int[] array = { 11, 22, 33 };
// java.lang.ArrayIndexOutOfBoundsException，数组的索引越界
array[4] = 44;
```

```
Object obj = "123.4";
// java.lang.ClassCastException，类型不匹配
Double d = (Double) obj;
```

小码哥教育 SEEMYGO

throws

- throws 的作用：将异常抛给上层方法

```
static void test() throws FileNotFoundException, ClassNotFoundException {
    PrintWriter out = new PrintWriter("F:/mj/520it.txt");
    Class cls = Class.forName("Dog");
}
```

- 如果 throws 后面的异常类型存在父子关系，保留父类型即可

```
static void test() throws Exception {
    PrintWriter out = new PrintWriter("F:/mj/520it.txt");
    Class cls = Class.forName("Dog");
}
```

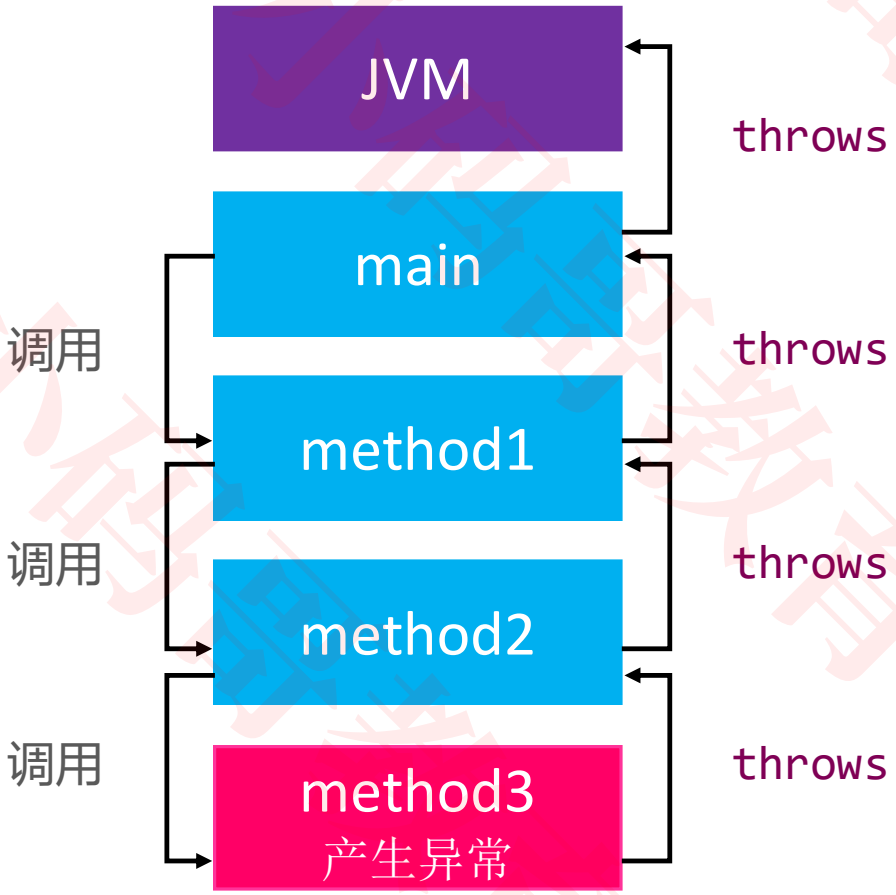
```
static void test() throws Throwable {
    PrintWriter out = new PrintWriter("F:/mj/520it.txt");
    Class cls = Class.forName("Dog");
}
```

throws 示例

```
static void test() throws FileNotFoundException {
    PrintWriter out = new PrintWriter("F:/mj/520it.txt");
    try {
        Class cls = Class.forName("Dog");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

- 可以一部分异常使用 try-catch 处理，另一部分异常使用 throws 处理

throws 的流程



- 如果异常最终抛给了 JVM
  - 那么整个 Java 程序将终止运行

throws 示例

```
static void method3() throws FileNotFoundException {
    PrintWriter out = new PrintWriter("F:/mj/520it.txt");
    out.print("My name is MJ.");
    out.close();
}
```

```
static void method2()
    throws FileNotFoundException {
    method3();
}
```

```
static void method1()
    throws FileNotFoundException {
    method2();
}
```

```
public static void main(String[] args) {
    try {
        method1();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

- 异常从 method3 方法中产生，一路向上抛
- 最后在 main 方法中使用 try-catch 处理了异常

throws 的细节

```
public class Person {
    public void test1() {}
    public void test2() throws IOException {}
    public void test3() throws IOException {}
    public void test4() throws IOException {}
}
```

```
public class Student extends Person {
    @Override
    public void test1() {}
    @Override
    public void test2() {}
    @Override
    public void test3() throws IOException {}
    @Override
    public void test4() throws FileNotFoundException{}
}
```

- 当父类的方法没有 throws 异常
  - 子类的重写方法也不能 throws 异常

- 当父类的方法有 throws 异常
  - 子类的重写方法可以
    - ✓ 不 throws 异常
    - ✓ throws 跟父类一样的异常
    - ✓ throws 父类异常的子类型

小码哥教育 SEEMYGO

throw

- 使用 throw 可以抛出一个新建的异常

```
public class Person {
    public Person(String name) throws Exception {
        if (name == null || name.length() == 0) {
            throw new Exception("name must not be empty.");
        }
    }
}
```

```
public class Person {
    public Person(String name) {
        if (name == null || name.length() == 0) {
            throw new IllegalArgumentException("name must not be empty.");
        }
    }
}
```

小码哥教育 SEEMYGO

自定义异常

- 开发中自定义的异常类型，基本都是以下 2 种做法

- 继承自 Exception
  - ✓ 使用起来代码会稍微复杂
  - ✓ 希望开发者重视这个异常、认真处理这个异常

- 继承自 RuntimeException
  - ✓ 使用起来代码会更加简洁
  - ✓ 不严格要求开发者去处理这个异常

```
public class EmptyNameException extends RuntimeException {
    public EmptyNameException() {
        super("name must not be empty.");
    }
}
```

```
public class WrongAgeException extends RuntimeException {
    private int age;
    public WrongAgeException(int age) {
        super("wrong age:" + age + ", age must be > 0");
    }
}
```

小码哥教育 SEEMYGO

自定义异常 - 示例

```
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        if (name == null || name.length() == 0) {
            throw new EmptyNameException();
        }
        if (age <= 0) {
            throw new WrongAgeException(age);
        }
        this.name = name;
        this.age = age;
    }
}
```

```
// WrongAgeException: wrong age:-10, age must be > 0
Person person = new Person("Jack", -10);
// 这句代码不会执行
System.out.println(1);
```

小码哥教育 SEEMYGO

使用异常的好处

- 将错误处理代码与普通代码区分开
- 能将错误信息传播到调用堆栈中
- 能对错误类型进行区分和分组

小码哥教育 @M了个J

小码哥教育 SEEMYGO

练习 – 编写一个断言类

```
public class Asserts {
    public static void test(boolean v) {
        if (v) return;
        System.err.println(new RuntimeException().getStackTrace()[1]);
    }
}
```

```
int age = 10;
Asserts.test(age > 0);

String name = "";
Asserts.test(name != null && name.length() != 0);
```