异常

内容提要

- 什么是异常
- 如何处理异常
  - try-catch-finally
  - throw
  - throws
  - 自定义异常
  - 异常链

什么是异常

- 异常字面翻译就是“意外、例外”的意思，也就是非正常情况。
- 异常本质上是程序上的错误。

程序中的异常

- 错误在我们编写程序的过程中会经常发生，包括编译期间和运行期间的错误。
- 在编译期间出现的错误有编译器帮助我们一起修正，然而运行期间的错误便不是编译器力所能及了，并且运行期间的错误往往是难以预料的。

程序中的异常

使用空的对象引用调用方法

```
String str=null;
System.out.println(str.length());
```

数组访问时下标越界

```
int[] ary={1,2,3};
for(int i=0;i<=3;i++){
    System.out.println(ary[i]);
}
```

算术运算时除数为0

```
int one =12;
int two=0;
System.out.println(one/two);
```

类型转换时无法正常转型

```
class Animal {

}

class Dog extends Animal {

}

class Cat extends Animal{

}

public class Test {
    public static void main(String[] args) {
        Animal a1 = new Dog();
        Animal a2 = new Cat();
        Dog d1 = (Dog)a1;
        Dog d2 = (Dog)a2;
    }
```

程序中的异常

- 在程序运行过程中，意外发生的情况，背离我们程序本身的意图的表现，都可以理解为异常。

- 当程序在运行期间出现了异常，如果置之不理，程序可能会不正常运行、强制中断运行、造成用户数据丢失、资源无法正常释放、直接导致系统崩溃，显然这不是我们希望看到的结果。

程序中的异常

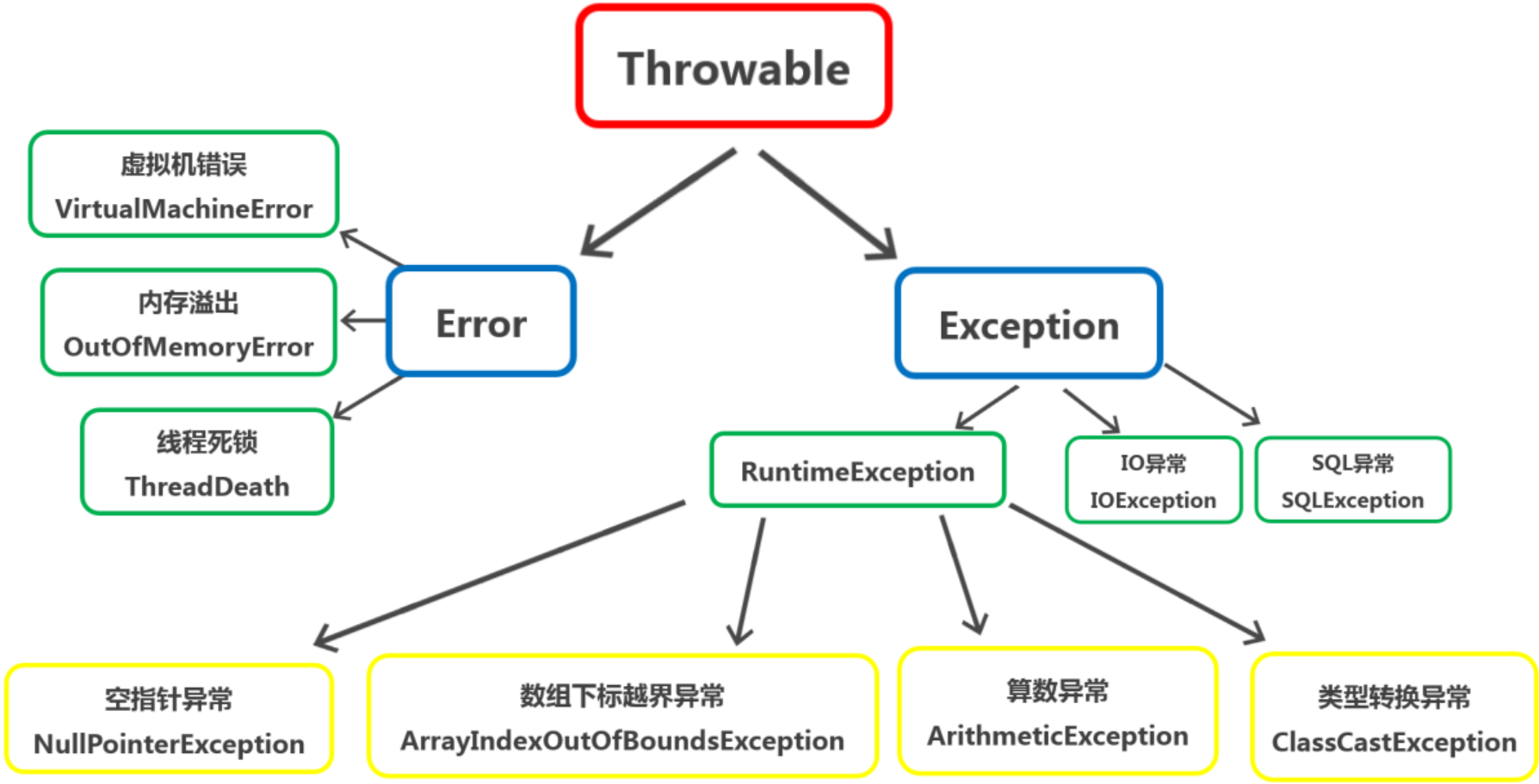
- 那么，如何针对程序运行期间产生的异常进行合理的处理？

- Java提供了异常机制来进行处理，通过异常机制，我们可以更好地提升程序的健壮性。

异常的分类

- 在程序开发中，异常指不期而至的各种状况。它是一个事件，当发生在程序运行期间时，会干扰正常的指令流程。

- 在Java中，通过Throwable及其子类描述各种不同的异常类型。


Throwable
Error
Exception
虚拟机错误
VirtualMachineError
内存溢出
OutOfMemoryError
线程死锁
ThreadDeath
RuntimeException
IO异常
IOException
SQL异常
SQLException
空指针异常
NullPointerException
数组下标越界异常
ArrayIndexOutOfBoundsException
算数异常
ArithmeticException
类型转换异常
ClassCastException

异常的分类

- Throwable有两个重要的子类：Exception 和 Error

Error

- Error是程序无法处理的错误，表示运行应用程序中较严重问题。大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM（Java 虚拟机）出现的问题。

- 例如，Java虚拟机运行错误（Virtual MachineError），当 JVM 不再有继续执行操作所需的内存资源时，将出现 OutOfMemoryError

Error

- 这些错误是不可查的，因为它们在应用程序的控制和处理能力之外，而且绝大多数是程序运行时不允许出现的状况。
- 对于设计合理的应用程序来说，即使确实发生了错误，本质上也不应该试图去处理它所引起的异常状况。
- 因此我们编写程序时不需要关心这类异常。

Exception

- Exception是程序本身可以处理的异常。异常处理通常指针对这种类型异常的处理。

- Exception类的异常包括 checked exception 和 unchecked exception。

unchecked exception

- unchecked exception：编译器不要求强制处置的异常。
- 包含RuntimeException类及其子类异常。
- 如NullPointerException(空指针异常)、IndexOutOfBoundsException(下标越界异常)等，这些异常是unchecked exception 。

unchecked exception

- Java编译器不会检查这些异常，在程序中可以选择捕获处理，也可以不处理，照样正常编译通过。

checked exception

- checked exception：编译器要求必须处置的异常。
- 是RuntimeException及其子类以外，其他的Exception类的子类。
- 如IOException、SQLException等

checked exception

- Java编译器会检查这些异常，当程序中可能出现这类异常时，要求必须进行异常处理，否则编译不会通过。

异常处理

- 在Java应用程序中，异常处理机制为：抛出异常、捕捉异常

抛出异常

- 当一个方法出现错误引发异常时，方法创建异常对象并交付运行时系统。
- 异常对象中包含了异常类型和异常出现时的程序状态等异常信息。
- 运行时系统负责寻找处置异常的代码并执行。

捕获异常

- 在方法抛出异常之后，运行时系统将转为寻找合适的异常处理器。
- 运行时系统从发生异常的方法开始，依次回查调用栈中的方法，当异常处理器所能处理的异常类型与方法抛出的异常类型相符时，即为合适的异常处理器。
- 当运行时系统遍历调用栈而未找到合适的异常处理器，则运行时系统终止。同时，意味着Java程序的终止。

捕获异常

- 对于运行时异常、错误或可查异常，Java技术所要求的异常处理方式有所不同。
- 总体来说，Java规定：对于可查异常必须捕捉、或者声明抛出。允许忽略不可查的RuntimeException和Error。
- 简单地说，异常总是先被抛出，后被捕捉的

异常处理

- 通过5个关键字来实现：try、catch、 finally、throw、throws

try-catch-finally

```
public void method(){
        try {
            // 代码段 1
            // 产生异常的代码段 2
        } catch (异常类型 ex) {
            // 对异常进行处理的代码段3
        }finally{
            // 代码段 4
        }
}
```

try-catch

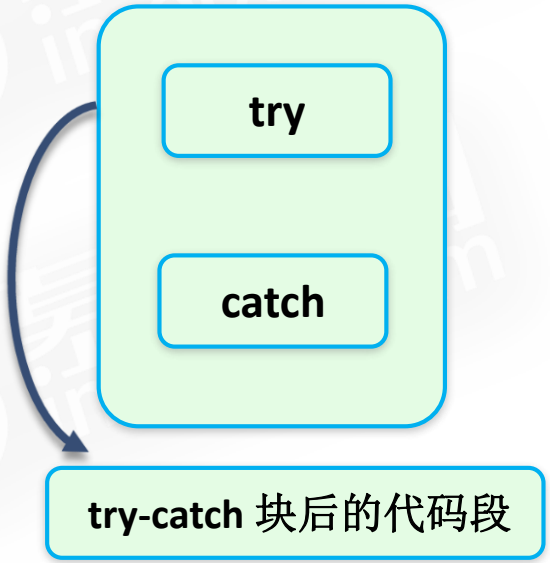
- 使用try-catch块捕获并处理异常

```
public void method(){
        try {
            // 代码段
        } catch (异常类型 ex) {
            // 对异常进行处理的代码段
        }
        // 代码段
}
```

try-catch

- 使用try-catch块捕获并处理异常—无异常

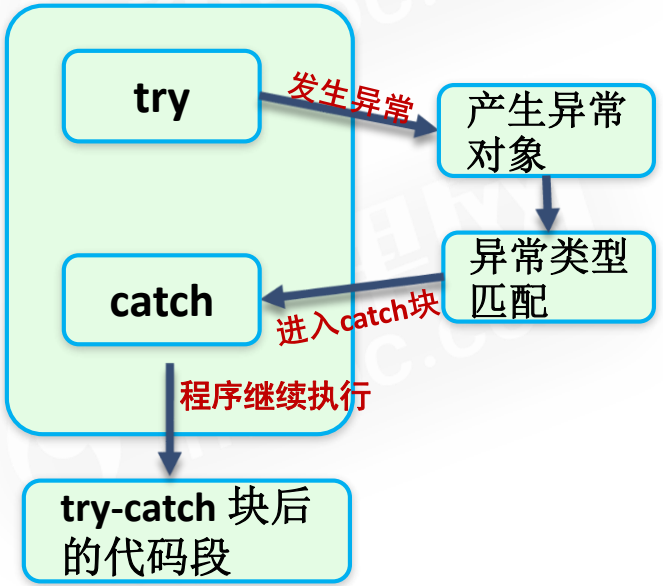
```
public void method(){
    try {
        // 代码段(此处不会产生异常)
    } catch (异常类型 ex) {
        // 对异常进行处理的代码段
    }
    // 代码段
}
```



try-catch

- 使用try-catch块捕获并处理异常—有异常并能正常匹配处理

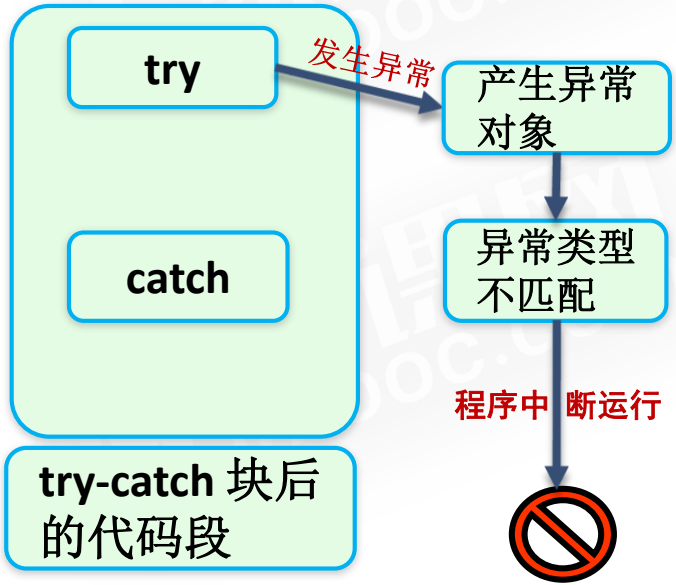
```
public void method(){
    try {
        // 代码段 1
        // 产生异常的代码段 2
        // 代码段 3
    } catch (异常类型 ex) {
        // 对异常进行处理的代码段4
    }
    // 代码段5
}
```



try-catch

- 使用try-catch块捕获并处理异常—有异常不能正常匹配处理

```
public void method(){
        try {
            // 代码段 1
            // 产生异常的代码段 2
            // 代码段 3
        } catch (异常类型 ex) {
            // 对异常进行处理的代码段4
        }
        // 代码段5
}
```



多重catch块

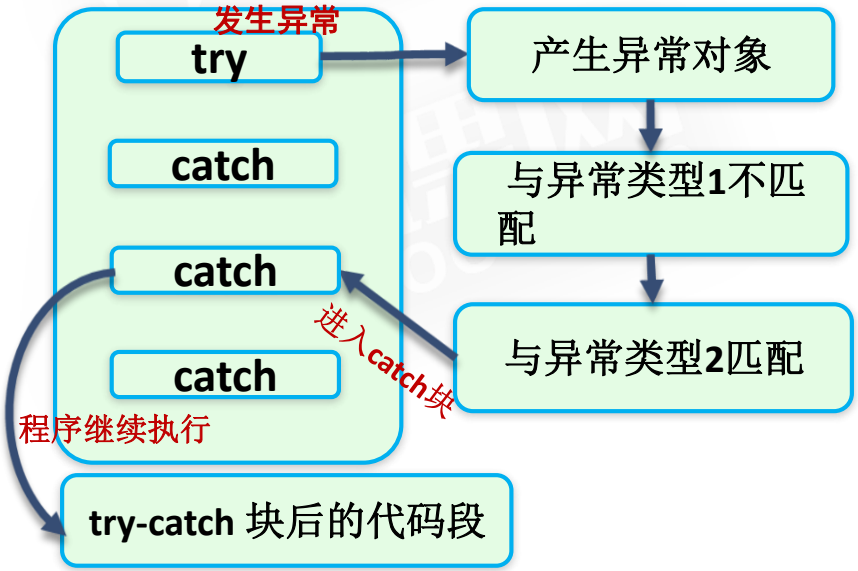
- 一旦某个catch捕获到匹配的异常类型，将进入异常处理代码。一经处理结束，就意味着整个try-catch语句结束。其他的catch子句不再有匹配和捕获异常类型的机会。

- 对于有多个catch子句的异常程序而言，应该尽量将捕获底层异常类的catch子 句放在前面，同时尽量将捕获相对高层的异常类的catch子句放在后面。否则，捕获底层异常类的catch子句将可能会被屏蔽。

多重catch块

- 引发多种类型的异常
  - 排列catch 语句的顺序：先子类后父类
  - 发生异常时按顺序逐个匹配
  - 只执行第一个与异常类型匹配的catch语句

```
public void method(){
        try {
            // 代码段
            // 产生异常(异常类型2)
        } catch (异常类型1 ex) {
            // 对异常进行处理的代码段
        } catch (异常类型2 ex) {
            // 对异常进行处理的代码段
        } catch (异常类型3 ex) {
            // 对异常进行处理的代码段
        }
        // 代码段
}
```



try-catch-finally

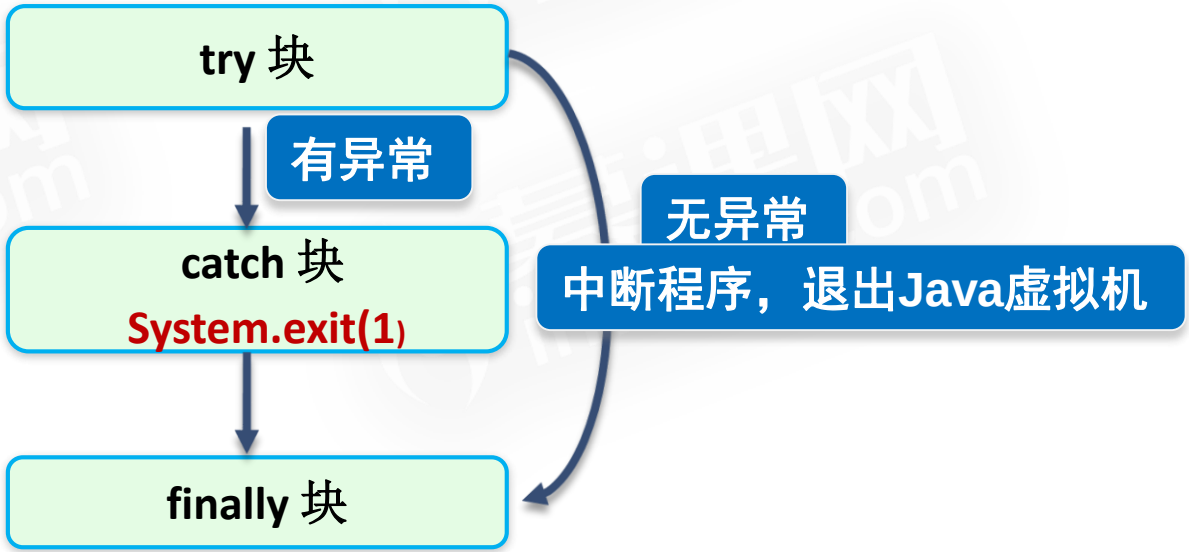
- try块后可以接零个或多个catch块
- 如果没有catch，则必须跟一个finally块

try-catch-finally

- catch、finally可选
- 语法组合：
  - try-catch
  - try-finally
  - try-catch-finally
  - try-catch-catch-finally

try-catch-finally

- 在try-catch块后加入finally块
  - 是否发生异常都执行
  - 不执行的唯一情况



try-catch-finally

```
public void method(){
        try {
            // 代码段 1
            // 产生异常的代码段 2
        } catch (异常类型 ex) {
            // 对异常进行处理的代码段3
            return;
        }finally{
            // 代码段 4
        }
}
```

常见的异常类型

异 常 类 型	说　　明
Exception	异常层次结构的父类
ArithmeticException	算术错误情形，如以零作除数
ArrayIndexOutOfBoundsException	数组下标越界
NullPointerException	尝试访问 null 对象成员
ClassNotFoundException	不能加载所需的类
IllegalArgumentException	方法接收到非法参数
ClassCastException	对象强制类型转换出错
NumberFormatException	数字格式转换异常，如把"abc"转换成数字

try-catch-finally

- **实际应用中的经验与总结**
  - **处理运行时异常时，采用逻辑去合理规避同时辅助try-catch处理**
  - **在多重catch块后面，可以加一个catch（Exception）来处理可能会被遗漏的异常**
  - **对于不确定的代码，也可以加上try-catch，处理潜在的异常**
  - **尽量去处理异常，切忌只是简单的调用printStackTrace()去打印输出**
  - **具体如何处理异常，要根据不同的业务需求和异常类型去决定**
  - **尽量添加finally语句块去释放占用的资源**

throw & throws

- 可以通过throws声明将要抛出何种类型的异常，通过throw将产生的异常抛出。

throws

- 如果一个方法可能会出现异常，但没有能力处理这种异常，可以在方法声明处用throws子句来声明抛出异常。

- 例如：汽车在运行时可能会出现故障，汽车本身没办法处理这个故障，那就让开车的人来处理。

throws

- **throws语句用在方法定义时声明该方法要抛出的异常类型。**

```
public void method() throws Exception1,Exception2,...,ExceptionN {
    // 可能产生异常的代码
}
```

- **当方法抛出异常列表中的异常时，方法将不对这些类型及其子类类型的异常作处理，而抛向调用该方法的方法，由他去处理。**

throws的使用规则

- 1、如果是不可查异常（unchecked exception），即Error、RuntimeException或它们的子类，那么可以不使用throws关键字来声明要抛出的异常，编译仍能顺利通过，但在运行时会被系统抛出。
- 2、如果一个方法中可能出现可查异常，要么用try-catch语句捕获，要么用throws子句声明将它抛出，否则会导致编译错误
- 3、当抛出了异常，则该方法的调用者必须处理或者重新抛出该异常。

throws的使用规则

- 4、当子类重写父类抛出异常的方法时，声明的异常必须是父类方法所声明异常的同类或子类。

throw

- throw用来抛出一个异常。
- 例如：throw new IOException();

- throw 抛出的只能够是可抛出类Throwable 或者其子类的实例对象。
- 例如：throw new String(“出错啦”)；是错误的

throw

```
public void method(){
    try {
        // 代码段 1
        throw new 异常类型( );
    } catch (异常类型 ex) {
      // 对异常进行处理的代码段2
    }
}
```

```
public void method() throws 异常类型{
    // 代码段 1
     throw new 异常类型( );
}
```

自定义异常

- 使用Java内置的异常类可以描述在编程时出现的大部分异常情况。
- 也可以通过自定义异常描述特定业务产生的异常类型。
- 所谓自定义异常，就是定义一个类，去继承Throwable类或者它的子类。

异常链

- 有时候我们会捕获一个异常后再抛出另一个异常

- 顾名思义就是：将异常发生的原因一个传一个串起来，即把底层的异常信息传给上层，这样逐层抛出