

# Java异常处理

天津卓讯科技有限公司



## 5 Java异常处理

- 5.1 异常的概念
- 5.2 异常的处理机制
- 5.3 异常的分类
- 5.4 异常的捕获和处理
- 5.5 自定义异常
- 5.6 Eclipse的debug

## 5.1 Java异常概念

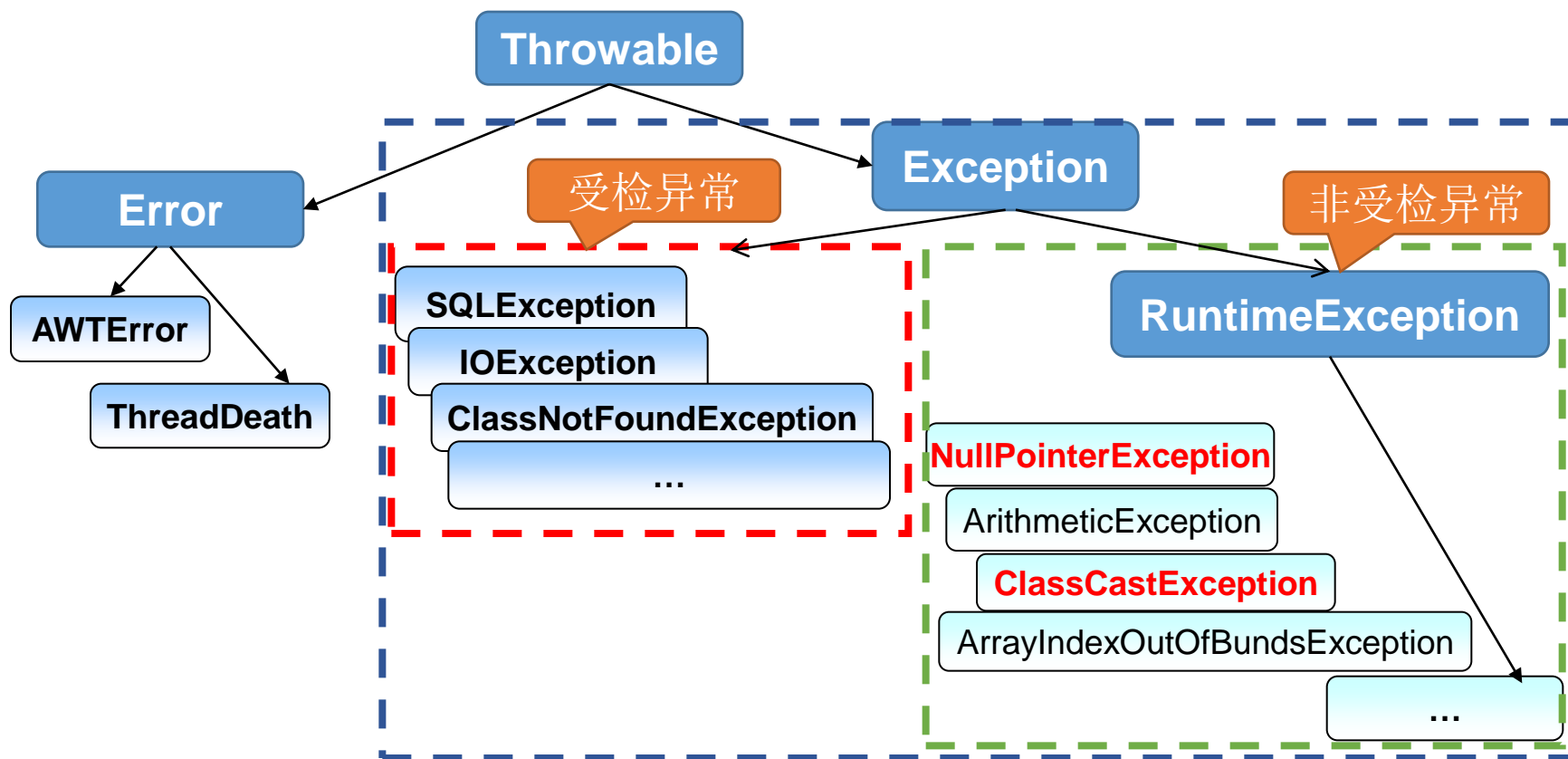
- 异常是程序在运行期发生的不正常的事件，它会打断指令的正常执行流程。
- 设计良好的程序应该在异常发生时提供处理这些不正常事件的方法，使程序不会因为异常的发生而阻断或产生不可预见的结果。
- Java语言使用异常处理机制为程序提供了异常处理的能力

```
public class TestException{  
    public static void main(String[] args){  
        String [] friends= {"lisa", "bily", "kessy"};  
        for(int i = 0; i < 5; i++) {  
            System.out.println(friends[i]);  
        }  
        System.out.println("程序结束");  
    }  
}
```

## 5.2 异常处理机制

- Java程序在执行过程中如果出现异常，会**自动生成一个异常类对象**，该异常对象将被自动提交给JVM，这个过程称为抛出(throw)异常。
- 当JVM接收到异常对象时，会寻找能处理这一异常的代码并把当前异常对象交给其处理，这一过程称为捕获(catch)异常和处理异常。
- 如果JVM找不到可以捕获异常的代码，则运行时系统将终止，相应的Java程序也将退出。

## 5.3 异常分类图



## 5.3 异常分类

- Java程序运行过程中所发生的异常事件从严重性可分为两类：
  - 错误(Error)：JVM系统内部错误或资源耗尽等严重情况 - 属于JVM需要负担的责任
    - 这一类异常事件无法恢复或不可能捕获，将导致应用程序中断。
  - 异常(Exception)：其它因为编程失误或偶然的外在因素导致的一般性问题。
    - 这类异常得到恰当的处理时，程序有机会恢复至正常运行状况。
- 程序员通常只能处理异常(Exception)，而对错误(Error)无能为力。

## 5.3 异常分类

- 从编程角度分：

- 非受检(unchecked)异常：编译器不要求强制处置的异常。
  - 一般是指编程时的逻辑失误。是程序员应该积极避免其出现的异常
  - java.lang.RuntimeException及它的子类都是非受检异常：
    - 错误的类型转换：java.lang.ClassCastException
    - 数组下标越界：java.lang.ArrayIndexOutOfBoundsException
    - 空指针访问：**java.lang.NullPointerException**
    - 算术异常(除0溢出)：java.lang.ArithmeticException
- 受检(checked)异常：编译器要求必须处置的异常。指的是程序在运行时由于外界因素造成的一般性异常。
  - 没有找到指定名称的类：java.lang.ClassNotFoundException
  - 访问不存在的文件：java.io.FileNotFoundException
  - 操作文件时发生的异常：java.io.IOException
  - 操作数据库时发生的异常：java.sql.SQLException
  - 网络操作时发生的异常：java.net.SocketException

```
public class TestException{
    public static void main(String[] args){
        String [] friends= {"lisa", "bily", "kessy"};
        try{
            for(int i = 0; i < 5; i++) {
                System.out.println(friends[i]);
            }
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("发生异常,请稍后再试...");
        }
    }
}
```



```
try{  
    ..... //可能产生异常的代码  
}catch(ExceptionName1 e ){  
    ..... //异常的处理代码  
}catch(ExceptionName2 e ){  
    ..... //异常的处理代码  
} [ finally{  
    .....//无论如何都会执行的语句  
} ]
```

- try 代码段包含的是可能产生异常的代码
- try 代码段后跟一个或多个catch代码段。(或跟一个finally代码段)
- 每个catch代码段只声明一种其能处理的特定类型的异常，并提供处理的方法。
- 当异常发生时，程序会中止当前的流程去执行相应的catch代码段。
- finally段的代码无论是否发生异常都执行。

- try语句

- try{ ... }语句包含了一段代码，这段代码就是一次捕获并处理异常的范围。
- 在执行过程中，该段代码可能会产生并抛出一种或几种类型的异常对象，它后面的catch语句要分别对这些异常做相应的处理。

## •catch语句

- 在catch语句块中是对异常进行处理的代码，每个try语句块可以伴随一个或多个catch语句，用于处理可能产生的不同类型的异常对象。
- 在catch中声明的异常对象（`catch(ExceptionName e)`）封装了异常事件发生的信息，在catch语句块中可以使用这个对象的一些方法获取这些信息。如：
  - `getMessage()` 方法，用来获得有关异常事件的字符串信息。
  - `printStackTrace()` 方法，用来跟踪异常事件发生时执行堆栈的内容。
- 使用多重 catch 语句时，异常子类一定要位于异常父类之前。
- 如果没有异常产生，所有的catch段的代码都会被忽略不执行。
- 受检异常必须try和catch

- finally

- finally语句为异常处理提供一个统一的出口，使得在控制流转到程序的其它部分以前，能够对程序的状态作统一的管理。
- 无论在try代码块中是否发生了异常事件，finally块中的语句都会被执行。
- 通常在finally语句块中可以进行资源的清理工作，如：
  - 关闭打开的文件
  - 删除临时文件
  - 关闭数据库的连接等
- finally语句是可选的。

```
public class TestException2{
    public int calculate(int num1, int num2) {
        int result = num1 / num2;
        return result;
    }
    public static void main(String[] args){
        TestException2 test = new TestException2();
        try{
            int i = test.calculate(100, 10);
            System.out.println(i);
        }catch(Exception e){
            System.out.println("父类异常..");
        }catch(ArithmeticException e){
            // 错误-不能到达的代码
            Sysetm.out.println("出异常啦");
            e.printStackTrace();
        }finally{
            System.out.println("finally语句块是始终要执行的");
        }
    }
}
```

- throw关键字用在方法代码中**主动抛出一个异常**。
  - 如果方法代码中自行抛出的异常是受检异常，则这个方法要用throws关键字声明这个异常。
- throws用来**声明**一个方法**可能会抛出的所有异常**。跟在方法签名的后面。
  - 如果一个方法声明的是受检异常，则在调用这个方法之处必须处置这个异常（谁调用谁处理）
    - 继续用throws向上声明。
- 注：重写一个方法时，它所声明的异常范围不能被扩大。

```
import java.io.*;
import java.sql.*;
class E {
    public String[] createArray(int length) {    //error!!
        if (length < 0) {        throw new Exception("数组长度小于0,不合法");
        } else {                return new String[length];        }
    }
    public void test() {        createArray(10);    }
    public void readFile() throws IOException,SQLException { }
}
/*class EE extends E {
    public void readFile() throws Exception { }
}*/
public class TestException3 {
    public static void main(String[] args) { ... }
}
```

## 5.5 自定义异常

- 创建自定义异常
- 继承自Exception 或其子类。
- 继承自RuntimeException或其子类。

```
public class MyException extends Exception {  
    public MyException() {    super(); }  
    public MyException(String msg) {    super(msg); }  
    public MyException(Throwable cause) {    super(cause); }  
    public MyException(String msg, Throwable cause) {    super(msg, cause); }  
}
```

- 使用自定义异常

```
public String[] createArray(int length) throws MyException {  
    if (length < 0) {  
        throw new MyException("数组长度小于0,不合法");  
    }  
    return new String[length];  
}
```



- 观察抛出的异常的名字和行号很重要。
- 调用内置类库中某个类的方法前，阅读其API文档了解它可能会抛出的异常。然后再据此决定自己是应该处理这些异常还是将其加入throws列表。
  - 应捕获和处理那些已知如何处理的异常，而传递那些不知如何处理的异常。
- 尽量减小try语句块的体积。
- 在处理异常时，应该打印出该异常的堆栈信息以方便调试。

```
public class TestEx {  
    public static int test(int x){  
        int i = 1;  
        try{  
            System.out.println("try块中10/x之前");  
            i = 10 / x;  
            System.out.println("try块中10/x之后");  
            return i;  
        }catch(Exception e){i = 100; System.out.println("catch块中");  
        }finally{ i = 1000; System.out.println("finally块");}  
        return i;  
    }  
    public static void main(String [] args){  
        System.out.println(TestEx.test(1));  
        System.out.println(TestEx.test(0));  
    }  
}
```

- 异常分类图
- try、catch、finally
- throw、throws
- 自定义异常