第十六章 异常

1. 异常的概述

1.1 没有异常存在的问题

代码块

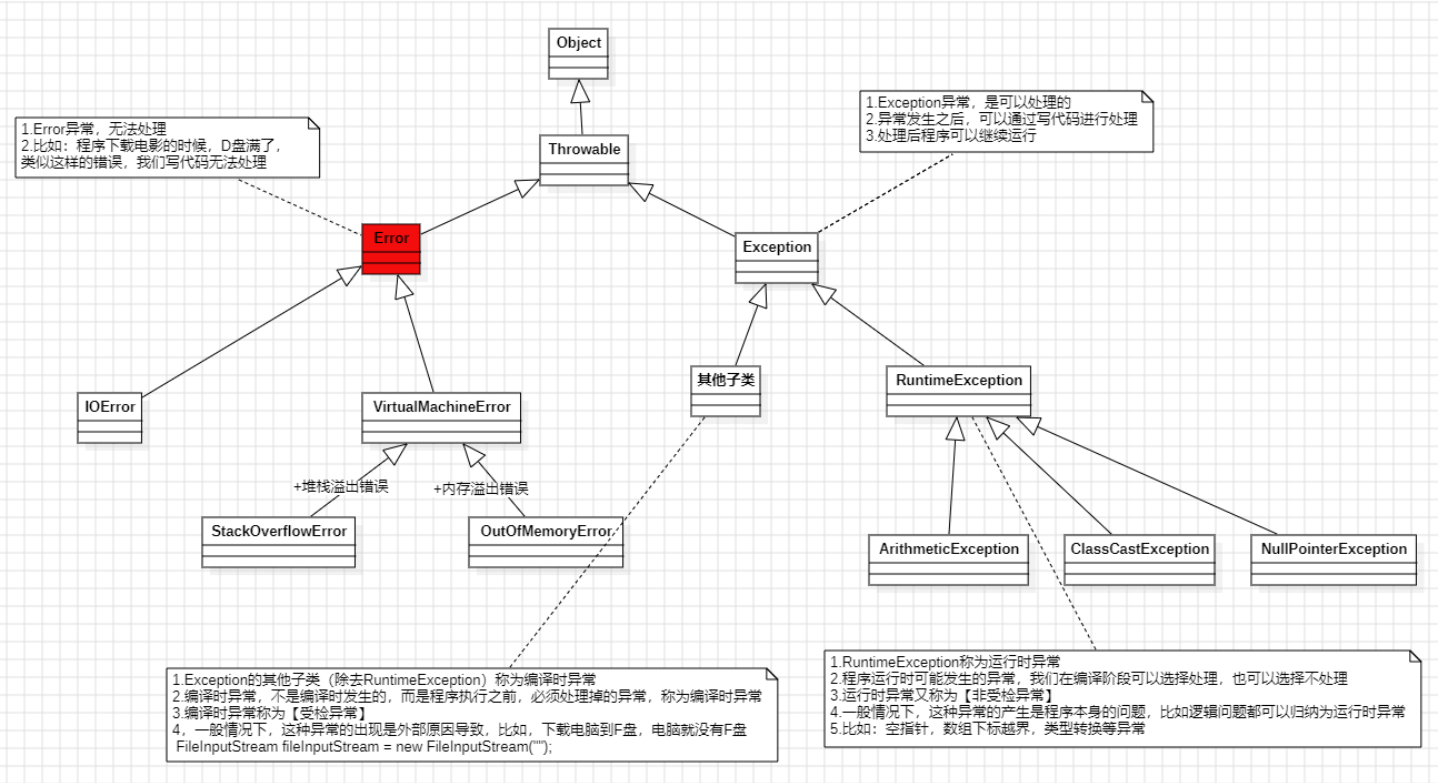
```
package com.powernode.exception06;

import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        int x = new Scanner(System.in).nextInt();
        int y = new Scanner(System.in).nextInt();
        /**
         * 1.没有异常存在的问题
         *   1.如果y = 0 （除数不能为0）
         *   2.程序出现异常，JVM终止工作
         *   3.为了让用户有更好的体验，可以通过异常处理，不让JVM直接终止工作
         */
        int num = 0;
        try {
            num = div(x, y);
        } catch (Exception e) {
            System.out.println("除数不能为0");
            return;
        }
        System.out.println(num);
    }

    private static int div(int x, int y) {
        return x / y;
    }

}
```

1.2 异常的概述

1. **异常是：程序运行的过程中发生的意外或者错误，（y=0）**

2. 异常处理：当程序发生意外或者错误时，对其做好标记，让程序继续执行，不至于JVM直接终止工作
3. 不同的异常对象，封装了不同的异常信息

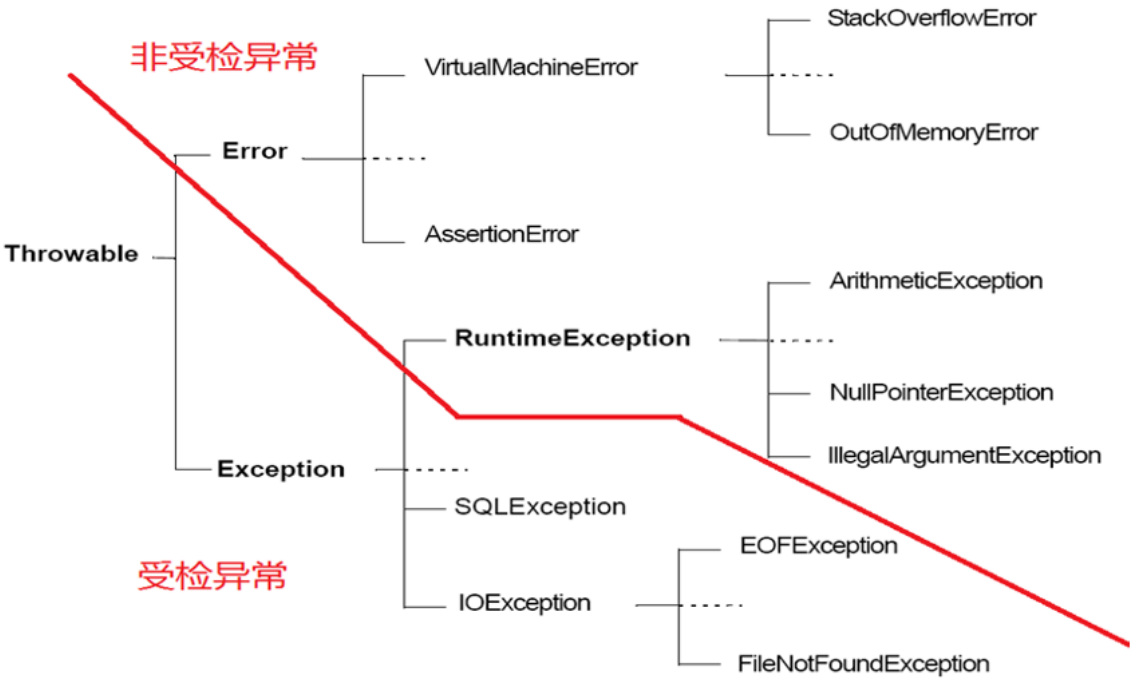
1.3 异常的家族体系结构



1.4 异常家族体系结构总结

1. 异常的继承关系
    a. 所有的类都继承了Throwable
    b. Throwable有两个重要的子类
        i. Error
        ii. Exception
            1. RuntimeException
            2. 其他子类
2. 异常的分类（重点）
    a. 非受检异常（运行时异常）
        i. 可以写代码进行处理，也可以不写
        ii. RuntimeException和Error的子类
    b. 受检异常（编译时异常）
        i. 必须写代码进行处理的

ii. Exception的子类（除去RuntimeException）



2. 异常的抛出机制

代码块

```
1  package com.powernode.exception06;
2
3  public class Test03 {
4      public static void main(String[] args) {
5          method01();
6      }
7      private static void method01() {
8          System.out.println("Test03.method01");
9          method02();
10     }
11     private static void method02() {
12         System.out.println("Test03.method02");
13         method03();
14     }
15     private static void method03() {
16         System.out.println("Test03.method03");
17         div(5, 0);
18     }
19     private static void div(int x, int y) {
20         System.out.println(x / y );
21     }
22 }
```

```java
public class Test03 {
    public static void main(String[] args) {
        method01();
    }
    private static void method01() {  1 usage
        System.out.println("Test03.method01");
        method02();
    }
    private static void method02() {  1 usage
        System.out.println("Test03.method02");
        method03();
    }
    private static void method03() {  1 usage
        System.out.println("Test03.method03");
        div( x: 5,  y: 0);
    }
    private static void div(int x, int y) {  1 usage
        System.out.println(x / y );
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException Create b
    at com.powernode.exception06.Test03.div(Test03.java:25)
    at com.powernode.exception06.Test03.method03(Test03.java:21
    at com.powernode.exception06.Test03.method02(Test03.java:16
    at com.powernode.exception06.Test03.method01(Test03.java:11
    at com.powernode.exception06.Test03.main(Test03.java:6)
```

发现：程序调用方法的顺序和抛出异常的顺序是相反的
因为方法调用时在栈中完成
1.栈： 先进后出，后进先出
2.队列：先进先出，后进后出

3. 异常处理方式

3.1 捕获异常并处理

3.1.1 try-catch捕获原理

代码块

```java
package com.powernode.exception07;

public class Test01 {
    public static void main(String[] args) {
        int x = 10;
        int y = 0;
        /**
         * 语法:
         *    try{
         *        可能发生异常的代码
         *    }catch(异常类 异常对象){
         *        异常处理：比如输出一句友好提示
         *    }
         */
        try{
            //y = 0 , 会发生
            //JVM自动创建一个异常对象，new ArithmeticException()
            int num = div(x,y);
```

```
19              //catch是怎么捕获到异常：JVM创建的异常对象，才catch中的对象赋值，如果赋值成功就可以捕获到
20          }catch(ArithmeticException e){//ArithmeticException e = new ArithmeticException()
21              System.out.println("除数不能为0");
22          }
23      }
24
25      private static int div(int x, int y) {
26          return x / y;
27      }
28  }
```

3.1.2 多个catch块处理顺序

代码块

```
1   package com.powernode.exception07;
2
3   import java.util.Scanner;
4
5   public class Test02 {
6       public static void main(String[] args) {
7
8           /**
9            * 语法：
10           *    try{
11           *        可能发生异常的代码
12           *    }catch(异常类 异常对象){
13           *        异常处理：比如输出一句友好提示
14           *    }catch(异常类 异常对象){
15           *        异常处理：比如输出一句友好提示
16           *    }
17           *
18           *    catch执行（根据出现的异常决定的）
19           *       1.从上到下依次执行
20           *       2.上面捕获到了异常，就不会往下执行
21           *       3.所以说:前面的异常范围 <=后面的异常范围
22           */
23          try{
24              //NumberFormatException ,ArithmeticException
25              int x = Integer.parseInt(args[0]);//接收字符串转换为整数，不需要掌握
26              int y = Integer.parseInt(args[1]);
27
28              int num = div(x ,y);
29              System.out.println(num);
```

```
30
31          }catch(ArithmeticException e){//ArithmeticException e = new
    ArithmeticException()
32              System.out.println("除数不能为0");
33          }catch (NumberFormatException e){
34              System.out.println("字符串无法转换为整数");
35          }
36
37      }
38
39      private static int div(int x, int y) {
40          return x / y;
41      }
42  }
```

3.1.3 catch块合并

代码块

```
1   package com.powernode.exception07;
2
3   public class Test03 {
4       public static void main(String[] args) {
5
6
7           try{
8               int x = Integer.parseInt(args[0]);
9               int y = Integer.parseInt(args[1]);
10
11              int num = div(x ,y);
12              System.out.println(num);
13
14          }catch(ArithmeticException | NumberFormatException e){
15              System.out.println("异常信息: " + e.getMessage());
16          }
17
18      }
19
20      private static int div(int x, int y) {
21          return x / y;
22      }
23  }
```

3.1.4 try-catch-finally

代码块

```
package com.powernode.exception07;

public class Test04 {
    public static void main(String[] args) {

        /**
         * 语法:
         *    try{
         *        可能发生异常的代码
         *    }catch(异常类 异常对象){
         *        异常处理:比如输出一句友好提示
         *    }
         *    ...
         *    finally{
         *        最终会中的代码
         *    }
         */
        try{
            int x = 10;
            int y = 1;
            int num = div(x ,y);
        }catch(ArithmeticException e){
            System.out.println("异常信息:" + e.getMessage());
        }finally {
            System.out.println("无论try执行,还是catch执行,finally都会执行");
        }

    }

    private static int div(int x, int y) {
        return x / y;
    }
}
```

3.1.5 try-finally

代码块

```
package com.powernode.exception07;

import java.util.concurrent.locks.ReentrantLock;

```

```java
public class Test05 {
    public static void main(String[] args) {

        /**
         * 语法:
         *    try{
         *        可能发生异常的代码
         *    }finally{
         *        最终会中的代码
         *    }
         *    在多线程场景中使用锁（如 ReentrantLock）时，finally 可确保锁被释放，避免死锁。
         */
        ReentrantLock lock = new ReentrantLock();
        try {
            lock.lock(); // 获取锁
            // 执行需要同步的操作（可能抛出异常）
        } finally {
            lock.unlock(); // 无论如何都释放锁
        }
    }
    /**
     * try-catch 可以组队出现
     * try-finally 可以组队出现
     * try-catch-finally 可以组队出现
     */


}
```

3.1.6 finally在return前执行

```
public class Test06 {
    public static void main(String[] args) {
        try {
            int num = div( x: 10, y: 0);
        } catch (ArithmeticException e) {
            System.out.println("除数不能为0");
            return;
        } finally {
            System.out.println("=====finally在return之前执行=====");
        }
    }

    private static int div(int x, int y) {  1 usage
        return x / y;
    }

}
```

1
finally执行后return才会执行
2
4
3

代码块

```
1   package com.powernode.exception07;
2
3   import java.util.concurrent.locks.ReentrantLock;
4
5   public class Test06 {
6       public static void main(String[] args) {
7           try {
8               int num = div(10,0);
9           } catch (ArithmeticException e) {
10              System.out.println("除数不能为0");
11              return;
12          } finally {
13              System.out.println("=====finally在return之前执行=====");
14          }
15      }
16
17      private static int div(int x, int y) {
18          return x / y;
19      }
20
21  }
```

3.1.7 exit,halt不会执行finally（了解）

代码块

```
package com.powernode.exception07;

public class Test07 {
    public static void main(String[] args) {
        try {
            int num = div(10,0);
        } catch (ArithmeticException e) {
            System.out.println("除数不能为0");
            //System.exit(0);//终止JVM,可以理解为关机，后面还可以处理资源
            Runtime.getRuntime().halt(0);//终止JVM，可以理解为断点，后面不可以处理资源
        } finally {
            System.out.println("=====finally不会执行=====");
        }
    }

    private static int div(int x, int y) {
        return x / y;
    }

}
```

3.2 抛出异常

3.2.1 方法声明处抛出

代码块

```
package com.powernode.exception08;

import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class Test01 {
    public static void main(String[] args) {
        try {
            fileReader();
        } catch (FileNotFoundException e) {
            System.out.println("文件没找到");
        }
    }
    //方法声明处抛出异常（自己无法处理，抛出给调用者）
    public static void fileReader()throws FileNotFoundException {
        FileInputStream fileInputStream = new FileInputStream("文件地址");
    }
}
```

3.2.2 方法内部throw手动抛出

代码块

```
package com.powernode.exception08;

public class Test02 {
    public static void main(String[] args) {
        div(5, 0);
    }

    private static void div(int x, int y) {
       /* try {
            System.out.println(x / y );
        } catch (ArithmeticException e) {
            throw new ArithmeticException();//上报
        }*/
        //先判断再执行的写法（第二种）效率更高，因为它避免了异常处理的额外开销。
        if (y == 0) {
            throw new ArithmeticException();//上报
        }
        System.out.println(x / y );
    }

}
```

4. 自定义异常

4.1 自定义异常（一）

1. 为什么要使用自定义异常
   a. 一个软件：有用户模块，有财务模块，报表模块
   b. 可以为摸个模块定义一个专有的异常，方便后期维护
2. 自定义异常的开发步骤：

代码块

```
1.受检异常
    1.自定义类，继承Exception
    2.定义构造器（可以根据实际情况选择几个参数的，一般情况全写）
2.非受检异常
    1.自定义类，继承了RuntimeException
```

```
6       2.定义构造器（可以根据实际情况选择几个参数的，一般情况全写）
```

代码块

```
1   package com.powernode.exception09;
2   class UserArithmeticException extends Exception{
3
4   }
5   public class Test01 {
6       public static void main(String[] args) {
7           try {
8               div(5, 0);
9           } catch (UserArithmeticException e) {
10              throw new RuntimeException(e);
11          }
12      }
13
14      private static void div(int x, int y)throws UserArithmeticException {
15          /*try {
16              System.out.println(x / y );
17          } catch (ArithmeticException e) {
18              throw new UserArithmeticException();//把系统的异常，转换为自定义异常
19          }*/
20          if (y == 0) {
21              throw new UserArithmeticException();
22          }
23          System.out.println(x / y);
24      }
25  }
```

4.2 自定义异常（二）

代码块

```
1   package com.powernode.exception10;
2   class UserArithmeticException extends Exception{
3       public UserArithmeticException(String message) {
4           super(message);
5       }
6
7       public UserArithmeticException(String message, Throwable cause) {
8           super(message, cause);
9       }
10
11      public UserArithmeticException(Throwable cause) {
```

```
12          super(cause);
13      }
14
15      public UserArithmeticException(String message, Throwable cause, boolean
    enableSuppression, boolean writableStackTrace) {
16          super(message, cause, enableSuppression, writableStackTrace);
17      }
18
19      public UserArithmeticException() {
20      }
21  }
22  public class Test {
23      public static void main(String[] args) {
24          try {
25              div(5, 0);
26          } catch (UserArithmeticException e) {
27              e.printStackTrace();//打印堆栈信息
28              System.out.println("异常信息：" + e.getMessage());
29          }
30      }
31
32      private static void div(int x, int y) throws UserArithmeticException {
33          try {
34              System.out.println(x / y );
35          } catch (ArithmeticException e) {
36              throw new UserArithmeticException("除数不能0",e);//把系统封装为自定义异
    常
37          }
38      }
39  }
```

4.3 方法重写中的异常

4.4 父类方法没有抛出异常

- 父类方法没有抛出异常，子类也不可以抛出异常（针对受检异常，非受检异常不受控制）

代码块

```
1   package com.powernode.exception11;
2   class Person{
3       public void eat(){
4           System.out.println("Person.eat");
5       }
6   }
```

```
class Teacher extends Person{
    //父类没有抛出异常，子类也不可以抛出异常（针对受检异常，非受检异常不受控制
    @Override
    public void eat()throws RuntimeException {
        System.out.println("Teacher.eat");
    }
}
public class Test {

}
```

4.5 父类方法有抛出异常

代码块

```
package com.powernode.exception12;

import java.io.FileNotFoundException;

class Person{
    public void eat()throws Exception{
        System.out.println("Person.eat");
    }
}
class Teacher extends Person{
    //子类抛出的异常 <= 父类抛出的异常，在实际工作中，一般都一样
    @Override
    public void eat()throws FileNotFoundException {
        System.out.println("Teacher.eat");
    }
}
public class Test {

}
```