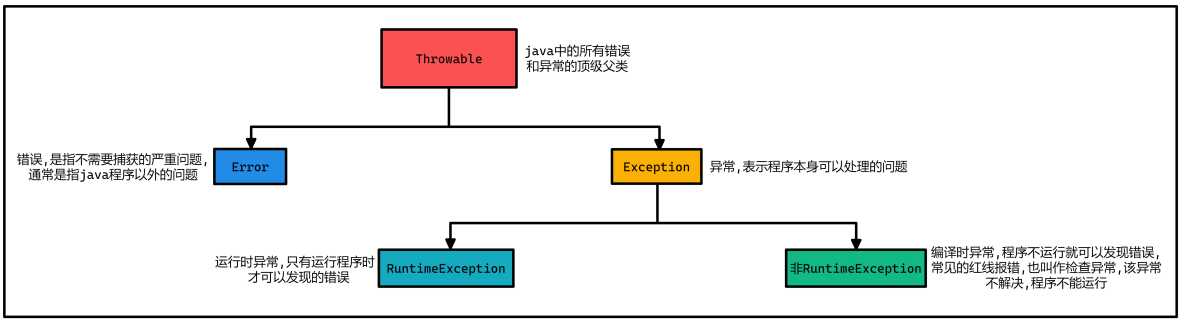
异常处理

1 概述

- **异常:** 就是指程序出现了不正常的情况
- 用来封装错误信息的对象。
- 组成结构：**类型，提示，行号**。

2 异常的继承结构



- Java中所有错误和异常的顶级父类是Throwable类
- Throwable类下有两个子类, 分别是Error和Exception, 两者的区别是:
  - Error: 是指不需要捕获的严重问题, 通常是java程序以外的问题, 比如硬件问题或者内存不足导致的问题等.因此, 如果java程序中出现了Error, 我们无需处理。
  - Exception: 称为异常类, 它表示程序本身可以处理的问题
- Exception下有很多异常子类, 其中有一个异常子类是RuntimeException类, 这里还可以将异常分为两大类:
  - 编译时异常:

    其他异常类以及不是RuntimeException子类的异常类都是检查异常(也叫编译时异常),在编写完程序后,Java编译器会对其进行检查, 如果检查出此类异常, 就必须要显式处理, 否则程序将无法进行编译。例如:ClassNotFoundException、FileNotFoundException、SQLException等都是编译时异常。
  - 运行时异常:

    RuntimeException以及子类被称为未经检查的异常(也叫运行时异常),这类异常通常在编写完程序后没有问题, 但是运行程序才出现异常, 需要我们回来修改代码进行解决的异常,这类异常无需显式处理, 当然也可以像编译时异常一样处理,例如:IndexOutOfBoundsException、ArithmeticException、NullPointerException、ClassCastException 等都是运行时异常。判断一个异常是不是运行时异常, 可以通过检查这个异常类是不是RuntimeException的子类, 或者检查这个异常是否只有在程序运行时才会出现!

3 虚拟机的默认处理方式

- 如果程序在运行时出现了问题，而我们又没有处理该问题，最终虚拟机会做默认的处理，而这种默认处理方式为:
  - 将异常的名称(类型)、异常的原因以及异常出现的位置等信息输出在了控制台（Console窗口）
  - 将程序停止运行（这意味着，出现异常的代码后面的代码将不会再执行）

3.1 异常示例1

```
package exception;

/**
 * 异常示例
 */
public class ExceptionDemo {
    public static void main(String[] args) {
        //没有报错,表示没有编译期异常
        //执行代码,报错了,说明有运行时异常
        int i = 1/0;
        System.out.println("hahaha");
    }
}
```

运行结果:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at exception.ExceptionDemo.main(ExceptionDemo.java:10)
```

3.2 异常示例2

```
package exception;

/**
 * 异常示例
 */
public class ExceptionDemo2 {
    public static void main(String[] args) {
        //定义一个数组
        int[] arr = {1,2,3,4,5,6};
        System.out.println(arr[0]);
        System.out.println(arr[1]);
        System.out.println(arr[2]);
        System.out.println(arr[3]);
        System.out.println(arr[4]);
        System.out.println(arr[5]);
        System.out.println(arr[6]);

    }
}
```

运行结果:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
    at exception.ExceptionDemo2.main(ExceptionDemo2.java:16)
```

- 如果程序出现了异常, 需要我们自己处理, 有两种方案:
  - 使用try...catch...进行处理(捕获异常)
  - 使用throws进行处理(抛出异常)

4 异常处理之try...catch...处理

4.1 语法格式

```
try{
  可能出现异常的代码;
}catch(异常类型 变量名){
  异常处理代码;
  //当try中的代码出现了异常并且这个异常能和catch中的异常类型匹配上，才会执行catch
  //反之，如果出现的异常和catch中的异常类型不匹配，就不会执行catch
}
```

执行流程

1. 程序执行到try{}中的代码时, 如果出现了异常, 将会自动产生一个异常对象, 该异常对象将会被提交给java运行时系统;
2. 当Java运行时系统接收到异常对象时, 回到catch()中寻找匹配的异常类型, 找到后就进入catch{}中进行异常的处理;
3. 执行完毕后, 程序还可以继续执行try...catch之后的代码

4.2 代码案例1

```
package cn.tedu.exception;

/**
 * 通过此案例学习异常捕获,了解基础语法结构
 * try{
 * 可能出现异常的代码;
 * }catch(异常类型 变量名){
 * 异常处理代码;
 * //当try中的代码出现了异常并且这个异常能和catch中的异常类型匹配上，才会执行catch
 * //反之，如果出现的异常和catch中的异常类型不匹配，就不会执行catch
 * }
 */
public class TryCatchDemo01 {
    public static void main(String[] args) {
        System.out.println("程序开始了!");
        //①将可能发生异常的代码放在try中
        try {
            String str = null;
            //在java中,null表示什么也没有,是不能调用方法的,所以虚拟机会提示空指针异常
            System.out.println(str.length());
        } catch (NullPointerException e) {//②通过catch关键字捕获try中代码出现的异常类型
            //③如果try中出现空指针异常,会捕获,并执行该catch中的内容
            System.out.println("服务器忙,请稍后重试!!!");
            //④将捕获的异常信息打印出来,一般是给程序员观看解决问题的
            e.printStackTrace();
        }
        System.out.println("程序结束了!");
    }
}
```

4.3 代码案例2

```
package cn.tedu.exception;

/**
 * 通过此案例学习异常捕获,异常可以捕获多个
 */
public class TryCatchDemo02 {
```

```
    public static void main(String[] args) {
        System.out.println("程序开始了!");
        try {
            String str = "";
            System.out.println(str.length());
            //char charAt(int index) 获取字符串中指定下标的字符
            System.out.println(str.charAt(0));
            //try中代码一旦发生异常,异常发生位置之后的代码不会被执行
            System.out.println("try中代码执行结束了!");
        } catch (NullPointerException e) {
            System.out.println("出现空指针异常!!!");
        } catch (StringIndexOutOfBoundsException e) {//catch可以定义多个
            System.out.println("出现了字符串下标越界异常!!!");
        }
        System.out.println("程序结束了!");
    }
}
```

4.4 代码案例3

```
package cn.tedu.exception;

/**
 * 通过此案例学习异常捕获,异常可以合并捕获
 */
public class TryCatchDemo03 {
    public static void main(String[] args) {
        System.out.println("程序开始了!");
        try {
            String str = null;
            System.out.println(str.length());
            System.out.println(str.charAt(0));
            System.out.println("try中代码执行结束了!");
        } catch (NullPointerException|StringIndexOutOfBoundsException e)
{//合并捕获
            System.out.println("出现空指针异常或者字符串下标越界异常!!!");
        }
        System.out.println("程序结束了!");
    }
}
```

4.5 代码案例4

```
package cn.tedu.exception;

/**
 * 通过此案例学习异常捕获,异常可以进行捕获父类异常进行兜底
 */
public class TryCatchDemo04 {
    public static void main(String[] args) {
        System.out.println("程序开始了!");
        try {
            String str = "a";
            System.out.println(str.length());
            System.out.println(str.charAt(0));
            //将字符串转换为整数,但是前提是字符串必须由数字组成
            System.out.println(Integer.parseInt(str));
            System.out.println("try中代码执行结束了!");
        } catch (NullPointerException | StringIndexOutOfBoundsException e)
{//合并捕获
            System.out.println("出现空指针异常或者字符串下标越界异常!!!");
```

```
        } catch (Exception e) {//捕获父类异常 工作中,一般情况下,依旧是出现一个异常写一个catch,不建议直接捕获Exception
            System.out.println("我也不知道出现什么异常,但是就是出现了异常!!!");
        }
        System.out.println("程序结束了!");
    }
}
```

5 finally块

- 作用就是确保一定要执行某些代码

5.1 代码案例

```
package cn.tedu.exception;

/**
 * finally块
 * finally是异常处理机制中的最后一块,可以直接跟在try语句块最后一个catch之后
 */
public class FinallyDemo {
    public static void main(String[] args) {
        System.out.println("程序开始了!");
        try {
            String str = "";
            System.out.println(str.length());
            //程序遇到return,直接跳出当前方法
            return;
        } catch (Exception e) {
            System.out.println("出现了一个错误");
        } finally {//一定会被执行
            System.out.println("必须要执行的内容~~");
        }
        System.out.println("程序结束了!");
    }
}
```

5.2 代码案例2

```
package cn.tedu.exception;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * 通过Finally代码块释放IO资源
 */
public class IOCloseDemo01 {
    public static void main(String[] args) {
        FileOutputStream fos = null;
        try {
            int a = 1 / 0;//模拟出问题
            fos = new FileOutputStream("demo/fos.txt");
            fos.write(1);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (fos != null) {
```

```
                    fos.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5.3 代码案例3

```
package cn.tedu.exception;

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * 释放IO流资源时,可以使用JDK7时推出的自动关闭特性
 */
public class IOCloseDemo02 {
    public static void main(String[] args) {
        //①在try关键字后,添加"()",将IO流对象的声明写入到这部分
        try (
                FileOutputStream fos = new
FileOutputStream("./demo/fos.txt");
        ) {
            //②关于流的使用代码则放在"{}"中
            fos.write(1);
            //③当try代码块结束,在"()"声明的流会自动释放资源
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

6 throw关键字

- 当程序发生错误而无法处理的时候,会**抛出**对应的异常对象
- 除此之外,在某些时刻,您可能会想要自行抛出异常,例如字异常处理结束后,再将异常抛出,让下一层异常处理块来捕捉,若想要自行抛出异常,您可以使用"throw"关键字,并生成执行的异常对象后抛出.
- 例如:throw new ArithmeticException();

6.1 Person

```
package cn.tedu.exception;

public class Person {
    private int age;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        //判断录入的年龄是否符合需求
        if (age < 0 || age > 100) {
            //此处可以抛异常
            throw new NullPointerException("您录入的年龄不合法!!");
```

```
        }
        this.age = age;
    }
}
```

6.2 ThrowDemo

```
package cn.tedu.exception;

/**
 * 模拟异常的抛出
 */
public class ThrowDemo {
    public static void main(String[] args) {
        Person p = new Person();
        //满足语法,但是不满业务需求
        p.setAge(10000);
        System.out.println("此人年龄:" + p.getAge() + "岁");
    }
}
```

7 异常处理之throws处理

- 程序中会声明许多的方法,这些方法中可能会因某些错误而引发异常,但是不希望直接在这个方法中处理这些异常,而希望调用这个它的方法来统一处理,这时候可以使用throws关键字来声明这个方法将会抛出的异常

7.1 语法格式

```
...方法名() throws 异常类名 {
  方法体
}
```

7.2 Person

```
package cn.tedu.exception;

public class Person {
    private int age;

    public int getAge() {
        return age;
    }

    /*
     * 如果要对外抛出异常,我们会使用throw关键字进行异常的抛出,
     * 如果抛出的是运行时异常,那么只能在运行过程中发现该异常
     * 而如果抛出的时编译时异常,那么必须要在方法的后面添加throws关键字
     * 告知调用者,在编译期时,需要检验该异常,所以在写代码时,就必须要处理该异常
     */
    public void setAge(int age) throws Exception {
        //判断录入的年龄是否符合需求
        if (age < 0 || age > 100) {
            //此处主动抛异常(表示抛出异常的动作)
            throw new Exception("您录入的年龄不合法!!");
        }
        this.age = age;
    }
}
```

7.3 ThrowDemo

```java
package cn.tedu.exception;

/**
 * 模拟异常的抛出
 */
public class ThrowDemo {
    public static void main(String[] args) {
        Person p = new Person();
        //满足语法,但是不满业务需求
        try {
            p.setAge(230);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("此人年龄:" + p.getAge() + "岁");
    }
}
```

8 自定义异常

8.1 IllegalAgeException

```java
package cn.tedu.exception;

/**
 * 自定义异常:非法的年龄异常
 * 自定义异常步骤:
 * ①类名要见名知意(一般是XxxException的格式,要求可以直观的表明异常的类型)
 * ②需要该类直接或者间接继承Exception类(表明当前类是异常子类)
 * ③提供父类中的所有的构造器
 */
public class IllegalAgeException extends Exception {
    public IllegalAgeException() {
    }

    public IllegalAgeException(String message) {
        super(message);
    }

    public IllegalAgeException(String message, Throwable cause) {
        super(message, cause);
    }

    public IllegalAgeException(Throwable cause) {
        super(cause);
    }

    public IllegalAgeException(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

8.2 Person

```java
package cn.tedu.exception;

public class Person {
    private int age;

    public int getAge() {
        return age;
    }

    /*
     * 如果要对外抛出异常,我们会使用throw关键字进行异常的抛出,
     * 如果抛出的是运行时异常,那么只能在运行过程中发现该异常
     * 而如果抛出的时编译时异常,那么必须要在方法的后面添加throws关键字
     * 告知调用者,在编译期时,需要检验该异常,所以在写代码时,就必须要处理该异常
     */
    public void setAge(int age) throws IllegalAgeException {
        //判断录入的年龄是否符合需求
        if (age < 0 || age > 100) {
            //此处主动抛异常(表示抛出异常的动作)
            throw new IllegalAgeException("您录入的年龄不合法!!");
        }
        this.age = age;
    }
}
```

8.3 ThrowDemo

```java
package cn.tedu.exception;

/**
 * 模拟异常的抛出
 */
public class ThrowDemo {
    public static void main(String[] args) {
        Person p = new Person();
        //满足语法,但是不满业务需求
        try {
            p.setAge(230);
        } catch (IllegalAgeException e) {
            e.printStackTrace();
        }
        System.out.println("此人年龄:" + p.getAge() + "岁");
    }
}
```

9 总结

什么时候需要try...catch异常, 什么时候需要throws异常?

1. 如果这个异常是方法内部的代码造成的异常, 而不是因为调用者的传参导致的异常(也就是说这个异常和调用者没有关系), 通常需要我们try...catch异常
2. 如果这个异常是调用者的传参导致的异常, 则将异常throws抛出(就是将异常抛给调用者)
3. 不要在main方法上throws抛出异常, 因为这样会将异常抛给虚拟机, 而虚拟机是不会帮我们处理异常的!(虚拟机会按照默认方式处理:输出异常信息以及终止程序执行!)