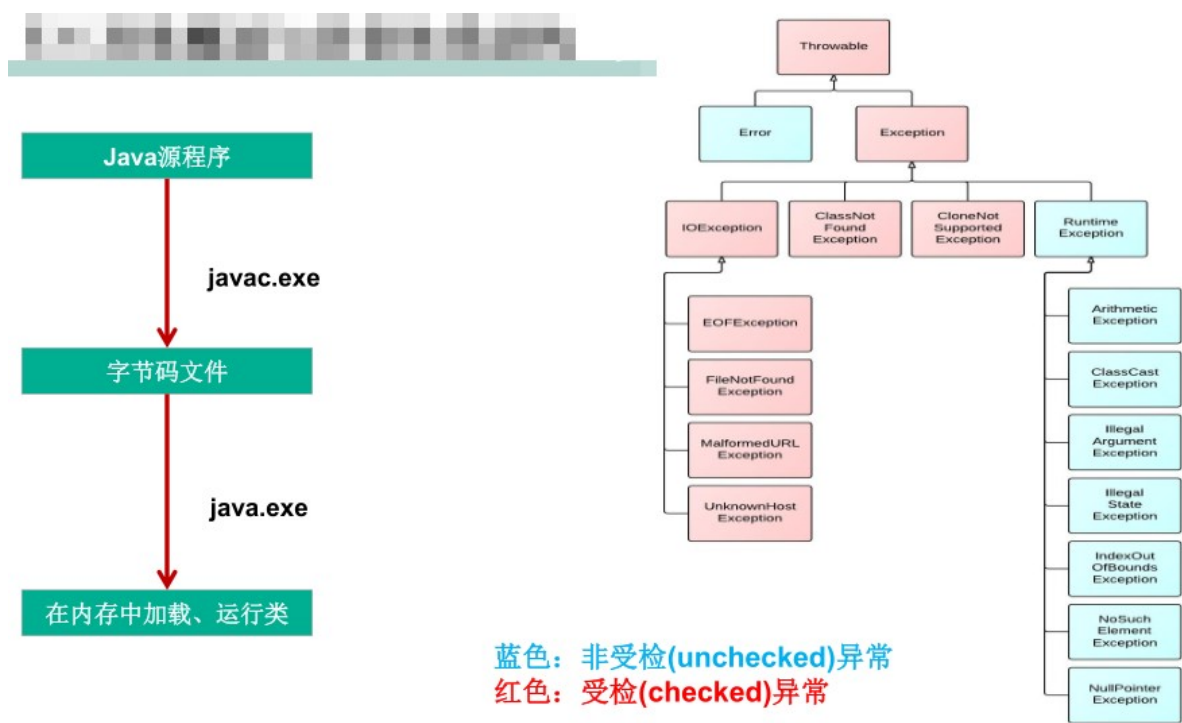


一、异常概述与异常体系结构

在使用计算机语言进行项目开发的过程中，即使程序员把代码写得**尽善尽美**，在系统的运行过程中仍然会遇到一些问题，因为很多问题不是靠代码能够避免的，比如：**客户输入数据的格式**，**读取文件是否存在**，**网络是否始终保持通畅**等等。

1、Error和Exception

- 异常：在Java语言中，将程序执行中发生的不正常情况称为“异常”。(开发过程中的语法错误和逻辑错误不是异常)
- Java程序在执行过程中所发生的异常事件可分为两类：
 - **Error**: Java虚拟机无法解决的严重问题。如：JVM系统内部错误、资源耗尽等严重情况。比如：**StackOverflowError**和**OOM**。一般不编写针对性的代码进行处理。
 - **Exception**: 其它因编程错误或偶然的外在因素导致的一般性问题，可以使用针对性的代码进行处理。例如：
 - ✓空指针访问
 - ✓试图读取不存在的文件
 - ✓网络连接中断
 - ✓数组角标越界
- 对于这些错误，一般有两种**解决方法**：一是遇到错误就终止程序的运行。另一种方法是由程序员在编写程序时，就考虑到错误的检测、错误消息的提示，以及错误的处理。
- 捕获错误最理想的是在**编译期间**，但有的错误只有在**运行时**才会发生。比如：**除数为0**，**数组下标越界**等
 - 分类：**编译时异常**和**运行时异常**



受检异常---》编译时异常

非受检异常---》运行时异常

2、运行时异常

- 是指编译器不要求强制处置的异常。一般是指编程时的逻辑错误，是程序员应该积极避免其出现的异常。**java.lang.RuntimeException**类及它的子类都是运行时异常。
- 对于这类异常，可以不作处理，因为这类异常很普遍，若全处理可能会对程序的可读性和运行效率产生影响。

3、编译时异常

- 是指编译器要求必须处置的异常。即程序在运行时由于外界因素造成的一般性异常。**编译器要求Java程序必须捕获或声明所有编译时异常。**
- 对于这类异常，如果程序不处理，可能会带来意想不到的结果。

二、常见异常

1、常见异常汇总

- java.lang.RuntimeException

- ClassCastException
- ArrayIndexOutOfBoundsException
- NullPointerException
- ArithmeticException
- NumberFormatException
- InputMismatchException
- . . .

- java.io.IOException

- FileNotFoundException
- EOFException

- `java.lang.ClassNotFoundException`

- `java.lang.InterruptedException`

- `java.io.FileNotFoundException`

- java.sql.SQLException

2、数组越界异常ArrayIndexOutOfBoundsException

```
public class IndexOutExp {
    public static void main(String[] args) {
        String friends[] = { "lisa", "bily", "kessy" };
        for (int i = 0; i < 5; i++) {
            System.out.println(friends[i]); // friends[4]?
        }
        System.out.println("\nthis is the end");
    }
}
```

程序IndexOutExp.java编译正确，运行结果：`java IndexOutExp`

```
lisa
bily
kessy
java.lang.ArrayIndexOutOfBoundsException
    at Test7_1.main(Test7_1.java:5)
Exception in thread "main"
```

3、空指针异常 NullPointerException

```

public class NullRef {
    int i = 1;

    public static void main(String[] args) {
        NullRef t = new NullRef();
        t = null;
        System.out.println(t.i);
    }
}

```

程序NullRef.java编译正确，运行结果：java NullRef

```

java.lang.NullPointerException
    at NullRef.main(NullRef.java:6)
Exception in thread "main"

```

4、 算术异常ArithmeticException

```

public class DivideZero {
    int x;

    public static void main(String[] args) {
        int y;
        DivideZero c=new DivideZero();
        y=3/c.x;
        System.out.println("program ends ok!");
    }
}

```

程序DivideZero.java编译正确，运行结果：java DivideZero

```

java.lang.ArithmeticException: / by zero
    at DivideZero.main(DivideZero.java:6)
Exception in thread "main"

```

5、 类型转换ClassCastException

```
public class Order {  
    public static void main(String[] args) {  
        Object obj = new Date();  
        Order order;  
        order = (Order) obj;  
        System.out.println(order);  
    }  
}
```

程序Person.java编译正确，运行结果：java Person

```
java.lang.java.lang.ClassCastException  
    at Person.main(Person.java:5)  
Exception in thread "main"
```

石子下海商

三、异常处理机制一：try-catch-finally

1、什么是异常处理

在编写程序时，经常要在可能出现错误的地方加上检测的代码，如进行 x/y 运算时，要检测分母为0，数据为空，输入的不是数据而是字符等。过多的if-else分支会导致程序的代码加长、臃肿，可读性差。因此采用异常处理机制。

Java异常处理

Java采用的异常处理机制，是将异常处理的程序代码集中在一起，与正常的程序代码分开，使得程序简洁、优雅，并易于维护。

Java异常处理的方式：

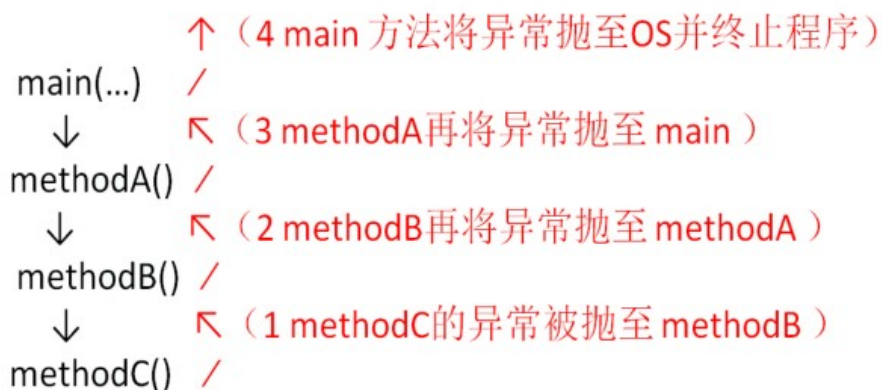
方式一：try-catch-finally

方式二：throws + 异常类型

2、异常处理机制描述

- Java提供的是异常处理的**抓抛模型**。
- Java程序的执行过程中如出现异常，会生成一个**异常类对象**，该异常对象将被提交给Java运行时系统，这个过程称为**抛出(throw)异常**。
- 异常对象的生成
 - 由虚拟机**自动生成**：程序运行过程中，虚拟机检测到程序发生了问题，如果在当前代码中没有找到相应的处理程序，就会在后台自动创建一个对应异常类的实例对象并抛出——自动抛出
 - 由开发人员**手动创建**：Exception exception = new ClassCastException();——创建好的异常对象不抛出对程序没有任何影响，和创建一个普通对象一样

异常的抛出机制



为保证程序正常执行，代码必须对可能出现的异常进行处理。

- 如果一个方法内抛出异常，该异常对象会被抛给调用者方法中处理。如果异常没有在调用者方法中处理，它继续被抛给这个调用方法的上层方法。这个过程将一直继续下去，直到异常被处理。这一过程称为**捕获(catch)异常**。
- 如果一个异常回到main()方法，并且main()也不处理，则程序运行终止。
- 程序员通常只能处理Exception，而对Error无能为力。

3、try-catch-finally语法

异常处理是通过try-catch-finally语句实现的。

```
try{
    ..... //可能产生异常的代码
}
catch( ExceptionName1 e ){
    ..... //当产生ExceptionName1型异常时的处置措施
}
catch( ExceptionName2 e ){
    ..... //当产生ExceptionName2型异常时的处置措施
}
[ finally{
    ..... //无论是否发生异常，都无条件执行的语句
} ]
```

●try

捕获异常的第一步是用try{...}语句块选定捕获异常的范围，将可能出现异常的代码放在try语句块中。

●catch (Exceptiontype e)

在catch语句块中是对异常对象进行处理的代码。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。

如果明确知道产生的是何种异常，可以用该异常类作为catch的参数；也可以用其父类作为catch的参数。

比如：可以用ArithmeticException类作为参数的地方，就可以用RuntimeException类作为参数，或者用所有异常的父类Exception类作为参数。但不能是与ArithmeticException类无关的异常，如NullPointerException（catch中的语句将不会执行）。

●捕获异常的有关信息：

与其它对象一样，可以访问一个异常对象的成员变量或调用它的方法。

➤getMessage() 获取异常信息，返回字符串

➤printStackTrace() 获取异常类名和异常信息，以及异常出现在程序中的位置。返回值void。

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.atguigu.exception.EcpTest.testException(EcpTest.java:29)
    at com.atguigu.exception.EcpTest.main(EcpTest.java:34)
```

异常名称 说明信息

↑ ↑

↓

堆栈信息

让天下没有难学的技术

●finally

- 捕获异常的最后一步是通过**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。
- 不论在**try**代码块中是否发生了异常事件，**catch**语句是否执行，**catch**语句是否有异常，**catch**语句中是否有**return**，**finally**块中的语句都会被执行。
- **finally**语句和**catch**语句是任选的

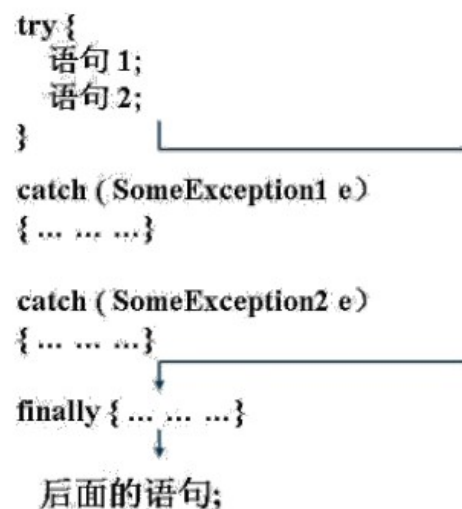
➤ 捕获SomeException2时:

```
try {  
    语句 1;  
    语句 2;  
}  
catch ( SomeException1 e )  
{ ..... }  
catch ( SomeException2 e )  
{ ... .. }  
finally { ... .. }  
后面的语句;
```



➤ 没有捕获到异常时:

```
try {  
    语句 1;  
    语句 2;  
}  
catch ( SomeException1 e )  
{ ... .. }  
catch ( SomeException2 e )  
{ ... .. }  
finally { ... .. }  
后面的语句;
```



- * 体会1: 使用**try-catch-finally**处理编译时异常，是得程序在编译时就不再报错，但是运行时仍可能报错。
- * 相当于我们使用**try-catch-finally**将一个编译时可能出现的异常，延迟到运行时出现。
- * 体会2: 开发中，由于运行时异常比较常见，所以我们通常就不针对运行时异常编写**try-catch-finally**了。
- * 针对于编译时异常，我们说一定要考虑异常的处理。

4、异常处理举例


```

public class IndexOutExp {
    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("index err");
        }
        System.out.println("\nthis is the end");
    }
}

```

程序IndexOutExp.java运行结果: **java IndexOutExp**
lisa
bily
kessy
index err
this is the end

让天下没有难题

```

public class DivideZero1 {
    int x;
    public static void main(String[] args) {
        int y;
        DivideZero1 c = new DivideZero1();
        try {
            y = 3 / c.x;
        } catch (ArithmeticException e) {
            System.out.println("divide by zero error!");
        }
        System.out.println("program ends ok!");
    }
}

```

程序DivideZero1运行结果: **java DivideZero1**
divide by zero error!
program ends ok!

让天下没有难题

5、不捕获异常时的情况

- 前面使用的异常都是**RuntimeException**类或是它的子类，这些类的异常的特点是：即使没有使用try和catch捕获，Java自己也能捕获，并且编译通过（但运行时会发生异常使得程序运行终止）。
- 如果抛出的异常是IOException等类型的非运行时异常，则**必须捕获，否则编译错误**。也就是说，我们必须处理编译时异常，将异常进行捕捉，转化为运行时异常

6、IOException 非运行时异常处理举例

```
import java.io.*;

public class IOExp {
    public static void main(String[] args) {
        FileInputStream in = new FileInputStream("atguigushk.txt");
        int b;
        b = in.read();
        while (b != -1) {
            System.out.print((char) b);
            b = in.read();
        }
        in.close();
    }
}
```

让天下没有难学的技术

IOException 异常处理举例(2)

```
import java.io.*;
public class IOExp {
    public static void main(String[] args) {
        try {
            FileInputStream in = new FileInputStream("atguigushk.txt");
            int b;
            b = in.read();
            while (b != -1) {
                System.out.print((char) b);
                b = in.read();
            }
            in.close();
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            System.out.println(" It's ok!");
        }
    }
}
```

让天下没有难学的技术

四、异常处理机制二：throws

1、throws 的使用

● 声明抛出异常是Java中处理异常的第二种方式

- 如果一个方法(中的语句执行时)可能生成某种异常，但是并不能确定如何处理这种异常，则此方法应显示地声明抛出异常，表明该方法将不对这些异常进行处理，而由该方法的调用者负责处理。
- 在方法声明中用 throws 语句可以声明抛出异常的列表，throws 后面的异常类型可以是方法中产生的异常类型，也可以是它的父类。

● 声明抛出异常举例：

```
public void readFile(String file) throws FileNotFoundException {
    .....
    // 读文件的操作可能产生FileNotFoundException类型的异常
    FileInputStream fis = new FileInputStream(file);
    .....
}
```

让天下没有难学的技术

```

import java.io.*;
public class ThrowsTest {
    public static void main(String[] args) {
        ThrowsTest t = new ThrowsTest();
        try {
            t.readFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void readFile() throws IOException {
        FileInputStream in = new FileInputStream("atguigushk.txt");
        int b;
        b = in.read();
        while (b != -1) {
            System.out.print((char) b);
            b = in.read();
        }
        in.close();
    }
}

```

让天下没有难学的。



2、重写方法声明抛出异常的原则

- 重写方法不能抛出比被重写方法范围更大的异常类型。在多态的情况下，对methodA()方法的调用-异常的捕获按父类声明的异常处理。

```

public class A {
    public void methodA() throws IOException {
        .....
    }
}
public class B1 extends A {
    public void methodA() throws FileNotFoundException {
        .....
    }
}
public class B2 extends A {
    public void methodA() throws Exception { //报错
        .....
    }
}

```

让天下没有难学的。


```

public class MyExpTest {
    public void regist(int num) throws MyException {
        if (num < 0)
            throw new MyException("人数为负值，不合理", 3);
        else
            System.out.println("登记人数" + num);
    }
    public void manager() {
        try {
            regist(100);
        } catch (MyException e) {
            System.out.print("登记失败，出错种类" + e.getId());
        }
        System.out.print("本次登记操作结束");
    }
    public static void main(String args[]) {
        MyExpTest t = new MyExpTest();
        t.manager();
    }
}

```

让天下

3、总结：异常处理5个关键字



例如：上游排污，下游治污 让天下没有难学的技

七、每日练习

1、运行时异常与一般异常有何异同

异常表示程序运行过程中可能出现的非正常状态。

运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。

java编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

2、Java中的异常处理机制的简单原理和应用？

当JAVA程序违反了JAVA的语义规则时，JAVA虚拟机就会将发生的错误表示为一个异常。

违反语义规则包括2种情况。一种是JAVA类库内置的语义检查。例如数组下标越界,会引发IndexOutOfBoundsException;访问null的对象时会引发NullPointerException。另一种情况就是JAVA允许程序员扩展这种语义检查,程序员可以创建自己的异常,并自由选择何时用throw关键字引发异常。所有的异常都是java.lang.Throwable的子类。

3、垃圾回收的优点和原理。并考虑2种回收机制？

Java语言中一个显著的特点就是引入了垃圾回收机制,使c++程序员最头疼的内存管理的问题迎刃而解,它使得Java程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制,Java中的对象不再有"作用域"的概念,只有对象的引用才有"作用域"。

垃圾回收可以有效的防止内存泄露,有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行,不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清楚和回收,程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。

回收机制有分代复制垃圾回收和标记垃圾回收,增量垃圾回收。

4、JAVA语言如何进行异常处理,关键字:throws,throw,try,catch,finally分别代表什么意义?在try块中可以抛出异常吗?

Java通过面向对象的方法进行异常处理,把各种不同的异常进行分类,并提供了良好的接口。在Java中,每个异常都是一个对象,它是Throwable类或其它子类的实例。当一个方法出现异常后便抛出一个异常对象,该对象中包含有异常信息,调用这个方法可以捕获到这个异常并进行处理。

Java的异常处理是通过5个关键词来实现的:try、catch、throw、throws和finally。一般情况下是用try来执行一段程序,如果出现异常,系统会抛出(throws)一个异常,这时候你可以通过它的类型来捕捉(catch)它,或最后(finally)由缺省处理器来处理。

用try来指定一块预防所有"异常"的程序。紧跟在try程序后面,应包含一个catch子句来指定你想要捕捉的"异常"的类型。

throw语句用来明确地抛出一个"异常"。

throws用来标明一个成员函数可能抛出的各种"异常"。

finally为确保一段代码不管发生什么"异常"都被执行一段代码。

可以在一个成员函数调用的外面写一个try语句,在这个成员函数内部写另一个try语句保护其他代码。每当遇到一个try语句,"异常"的框架就放到堆栈上面,直到所有的try语句都完成。如果下一级的try语句没有对某种"异常"进行处理,堆栈就会展开,直到遇到有处理这种"异常"的try语句。

5、try {}里有一个return语句,那么紧跟在这个try后的finally {}里的code会不会被执行,什么时候被执行,在return前还是后?

会执行,在return前执行

6、给我一个你最常见到的runtime exception?

常见的运行时异常有如下这些

ArithmeticException,

ArrayStoreException,

BufferOverflowException,
BufferUnderflowException,
CannotRedoException,
CannotUndoException,
ClassCastException,
CMMException,
ConcurrentModificationException,
DOMException,
EmptyStackException,
IllegalArgumentException,
IllegalMonitorStateException, I
IllegalPathStateException,
IllegalStateException,
ImagingOpException,
IndexOutOfBoundsException,
MissingResourceException,
NegativeArraySizeException,
NoSuchElementException,
NullPointerException,
ProfileDataException,
ProviderException,
RasterFormatException,
SecurityException,
SystemException,
UndeclaredThrowableException,
UnmodifiableSetException,
UnsupportedOperationException

7、error和exception有什么区别？

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。

exception 表示一种设计或实现问题。也就是说，它表示如果程序运行正常，从不会发生的情况。