一. 异常的基本概念

1. 异常产生的条件或者称为异常情况。在Java代码中哪些是异常情况呢？ 例如：

a. 整数相除运算中，分母为0;

b. 通过一个没有指向任何具体对象的引用去访问对象的方法；

c. 使用数组长度作为下标访问数组元素；

d. 将一个引用强制转化成不相干的对象；

等等；

2. 异常会改变正常程序流程;

异常产生后，正常的程序流程被打破了，要么程序中止，要么程序被转向异常处理的语句；

3. 当一个异常的事件发生后，该异常被虚拟机封装形成异常对象抛出。

4. 用来负责处理异常的代码被称为异常处理器

5. 通过异常处理器来捕获异常

二. try...catch语句

在Java语言中，用try...catch语句来捕获处理异常。格式如下：

try {

可能会出现异常情况的代码；业务实现代码；

} catch(异常类型 异常参数) {

异常处理代码

} catch(异常类型 异常参数) {

异常处理代码

}final{

一定会执行的代码

用于回收在try块打开的物理资源

}

1. 如果try代码块中没有抛出异常，try代码块中语句会顺序执行完，catch代码块内容不会被执行；

2. 如果try代码块中抛出catch代码块所声明的异常类型对象，程序跳过try代码块中接下来代码，直接执行catch代码块中对应内容;

a. 可以存在多个catch代码块，究竟执行哪个，看抛出的异常对象是否是catch代码块中异常类型；

b. 异常只能被一个异常处理器所处理, 不能声明两个异常处理器处理相同类型的异常；

c. 多个catch语句块所声明的异常类型不能越来越小；一定要先捕获小异常，再捕获大异常；

d. 不能捕获一个在try语句块中没有抛出的异常；

3. 如果try代码块中抛出catch代码块未声明的异常类型对象，异常被抛给调用者；哪个调用了这段语句块哪个负责处理这个异常；

4. 执行final中代码块

5. 异常处理可以嵌套，但通常没必要使用超过两层的嵌套异常处理

三. finally语句: 任何情况下都必须执行的代码

由于异常会强制中断正常流程，这会使得某些不管在任何情况下都必须执行的步骤被忽略，从而影响程序的健壮性。

Java的垃圾回收机制不会回收任何物理资源，只能回收堆内存中对象所占用的内存。除非在try块、catch块中调用了退出虚拟机的方法（如System.exit(1))，否则不管在try块、catch块中执行怎样的代码（如return），出现怎样的情况，异常处理的finally块总会被执行。尽量避免在finally块里使用return或throw等导致方法终止的语句，否则可能出现一些很奇怪的情况。

这在某些情况下是可行的，但不值得推荐：

　　 a. 把与try代码块相关的操作孤立开来，使程序结构松散，可读性差；

b. 影响程序的健壮性。假设catch语句块中继续有异常抛出，关门动作便不会执行。

四. 异常调用栈

异常处理时所经过的一系列方法调用过程被称为异常调用栈。

1. 异常的传播

哪个调用，哪个处理；

a. 异常情况发生后，发生异常所在的方法可以处理；

b. 异常所在的方法内部没有处理，该异常将被抛给该方法调用者，调用者可以处理；

　　　　c. 如调用者没有处理，异常将被继续抛出；如一直没有对异常处理，异常将被抛至虚拟机；

2. 如果异常没有被捕获，那么异常将使你的程序将被停止。

异常产生后，如果一直没有进行捕获处理，该异常被抛给虚拟机。程序将被终止。

3. 经常会使用的异常API

getMessage：获得具体的异常出错信息，可能为null。

printStatckTrace()：打印异常在传播过程中所经过的一系列方法的信息，简称异常处理方法调用栈信息；在程序调试阶段，此方法可用于跟踪错误。

printStatckTrace(PrintStream s)：将该异常的跟踪栈信息输出到指定输出流

五. 异常层级关系

所有异常类的祖先类为java.lang.Throwable类。它有两个直接的子类：

|--Error

| |--RuntimeException--|1~6

Throwable--|--Exception--|--IOException

|--SQLException

|--CheckedException

1. Error类：表示仅靠程序本身无法恢复的严重错误，比如内存空间不足，或者Java虚拟机的方法调用栈溢出。在大多数情况下，遇到这样的错误时，建议让程序终止。

2. Exception类：表示程序本身可以处理的异常。Exception还可以分为两种：运行时异常和受检查异常。

a. 运行时异常

1) 基本概念

RuntimeException类及其子类都被称为运行时异常，这种异常的特点是Java编译器不会检查它，也就是说，当程序中可能出现这类异常时，即使没有用try...catch语句捕获它，也没有用throws子句声明抛出它，还是会编译通过。

例如divide()方法的参数b为0,执行a/b操作时会出现ArithmeticException异常，它属于运行时异常，Java编译器不会检查它。

public int divide(int a, int b) {

return a/b; //当参数b为0, 抛出ArithmeticException

}

2) 深入解析

运行时异常表示无法让程序恢复运行的异常，导致这种异常的原因通常是由于执行了错误操作。一旦出现了错误操作，建议终止程序，因此Java编译器不检查这种异常。

运行时异常应该尽量避免。在程序调试阶段，遇到这种异常时，正确的做法是改进程序的设计和实现方式，修改程序中的错误，从而避免这种异常。捕获它并且使程序恢复运行并不是明智的办法。

3) 对比

与Error类相比：

相同点：

i. Java编译器都不会检查它们；

ii.当程序运行时出现它们， 都会终止程序；

不同点：

i. Error类及其子类表示的错误通常是由Java虚拟机抛出的，在JDK中预定义了一些错误类，比如

OutOfMemoryError和StackOutofMemoryError。

RuntimeException表示程序代码中的错误；

ii.Error类一般不会扩展来创建用户自定义的错误类；

RuntimeException是可以扩展的，用户可以根据特定的问题领域来创建相关的运行时异常类；

b. 受检查异常。

除了RuntimeException及其子类以外， 其他的Exception类及其子类都属于受检查异常(Checked Exception)。 这种异常的特点是Java编译器会检查它，也就是说，当程序中可能出现这类异常时，要么用try...catch语句捕获它，要么用throws子句声明抛出它，否则编译不会通过。

六. 一些未检查的异常RuntimeException

1. java.lang.ArithmeticException

算术异常 如：除0;

2. java.lang.NullPointerException

空指针引用 如：没初始化一个References便使用;

3. java.lang.ArrayIndexoutofBoundsException

数组越界 如：调用一个有十个元素的Array的第十一个元素的内容;

4. java.lang.ClassCastException

强制类型转换异常

5. java.lang.NumberFormatException

数据格式异常 如：Integer.parseInt("a");

6. java.lang.NegativeArraySizeException 数组长度为负数异常

七. 异常声明和处理

由于与业务需求不符而产生的异常，必须由程序员来决定抛出，系统无法抛出这种异常。

当JAVA运行时接收到开发者自行抛出的异常，同样会中止当前的执行流，跳到该异常对应的catch块，由该catch块来处理异常。

1. 自己主动使用throw语句的时候代码会抛出异常；

throw用于抛出一个实际的异常，可以单独作为语句使用。

2. 使用try-catch-finally语句结构处理或在方法声明上声明throws继续抛出；

throws在方法签名中使用，用于声明该方法可能抛出的异常。

catch和throw结合使用的情况在大型企业级应用中非常常用。

这种把捕获一个异常然后接着抛出另一个异常，并把原始异常信息保存下来的是一种典型的链式处理（23种设计模式之一：职责链模式），也被称为”异常链“。

虽然printStackTrace()方法可以很方便地用于追踪异常的发生情况，可以用它来调试程序，但在最后发布的程序中，应该避免使用它，而应该对捕获的异常进行适当的处理，而不是简单地将异常追踪栈信息打印出来。

user--->表现层：用户界面---->中间层：实现业务逻辑---->持久层：保存数据

异常处理语句的语法规则：

1. try代码块不能脱离catch代码块或finally代码块而单独存在。try代码块后面至少有一个catch代码块或finally代码块。

2. try代码块后面可以有零个或多个catch代码块，还可以有零个或至多一个finally代码块。如果catch代码块和finally代码块并存，finally代码块必须在catch代码块后面。

3. try代码块后面可以只跟finally代码块。

4. 在try代码块中定义的变量的作用域为try代码块，在catch代码块和finally代码块中不能访问该变量。

5. 当try代码块后面有多个catch代码块时，Java虚拟机会把实际抛出的异常对象依次和各个catch代码块声明的异常类型匹配，如果异常对象为某个异常或其子类的实例，就执行这个catch代码块，而不会再执行其他的catch代码块。

6. 如果一个方法可能出现受检查异常，要么用try...catch语句捕获，要么用throws子句声明将它抛出。

7. throw语句后面不允许紧跟其它语句，因为这些语句永远不会被执行。

异常处理的基本准则：

1.不要过度使用异常

异常处理机制的初衷是将不可预期异常的处理代码和正常业务逻辑处理代码分离，因此绝不要使用异常处理来代替正常的业务逻辑判断。不要使用异常处理来代替正常的流程控制。

2.不要使用过于庞大的try块

3.避免使用catch ALL预计

4.不要忽略捕获到的异常

异常处理目标：

1.使程序代码混乱最小化

2.捕获并保留诊断信息

3.通知合适的人员

4.采用合适的方式结束异常活动