[

Java千百问_04异常处理（001）_什么是java中的异常

,
点击进入_更多_Java千百问

1、java异常是什么

java在执行期间产生了某些问题，导致执行中断，这一问题就称为异常。
不同的原因都可能产生异常，包括以下内容：
==用户输入无效数据。
==需要打开的文件不存在。
==网络连接已丢失。
==JVM已经耗尽内存。
==将null当作某种对象进行操作。

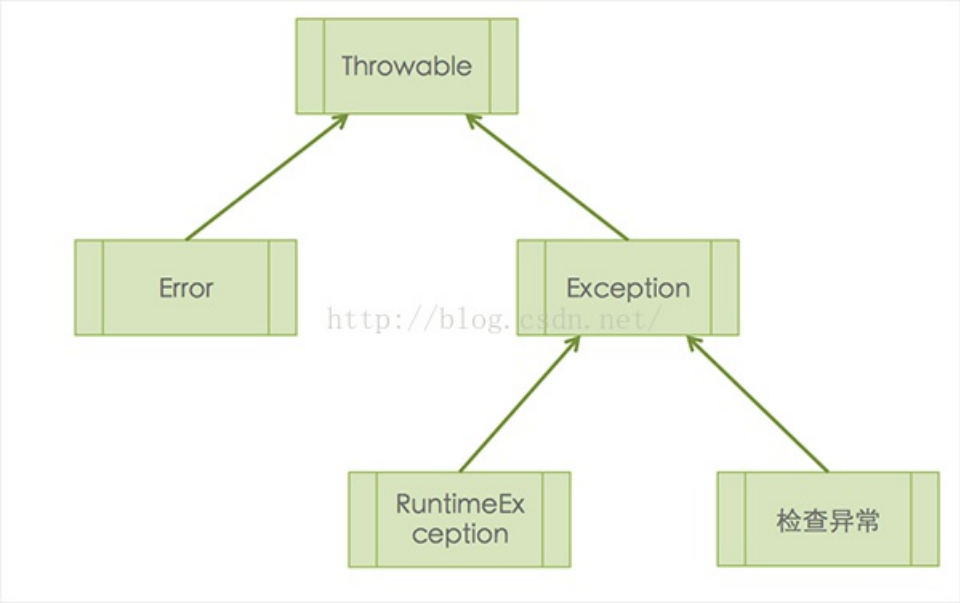
2、异常都分为哪些

要了解在Java中如何异常处理工作，需要了解三类异常：
检查异常（checked exception）
检查异常通常是用户错误，程序员并不可预见的问题。例如，如果一个文件被打开，但该文件无法找到，则会出现异常。这些异常并不能在编译时被发现。
运行时异常（runtime exception也叫unchecked exception）
运行时异常时本来可以由程序避免的异常。而不是已检查异常，运行时异常是在编译时被忽略。这里的运行时异常并不是我们所说的运行期间产生的异常，只是Java中用运行时异常这个术语来表示而已。另外，所有Exception异常都是在运行期间产生的。
错误（error）

无法处理的异常，比如OutOfMemoryError，一般发生这种异常，JVM会选择终止程序。因此我们编写程序时不需要关心这类异常。

要想自定义异常看这里：如何自定义异常

3、异常层次结构是怎样的



在Java中，所有异常类的父类是Throwable类，Error类是error类型异常的父类，Exception类是exception类型异常的父类，RuntimeException类是所有运行时异常的父类，RuntimeException以外的并且继承Exception的类是非运行时异常。

常见的RuntimeException包括NullPointerException、IndexOutOfBoundsException、IllegalArgumentException等。

常见的非RuntimeException包括IOException、SQLException等。

4、异常提供哪些方法

以下是Throwable类中比较重要的方法。

1 public String getMessage()

返回有关已发生异常的详细消息。此消息在Throwable的构造函数中被初始化。

2 public Throwable getCause()

返回异常由一个Throwable对象所表示的错误原因。

3 public String toString()

返回getMessage()结果的名称。

4 public void printStackTrace()

打印toString()结果以及堆栈跟踪信息到System.err，输出错误流。

5 public StackTraceElement [] getStackTrace()

返回堆栈跟踪信息的数组。索引为0的元素表示堆栈的顶部，最后一个元素表示堆栈的底部。

如何捕获java中异常看这里：如何捕获异常

如何将异常抛出看这里：如何抛出异常

]

[

Java千百问_04异常处理（002）_java如何捕获异常

,

点击进入_更多_Java千百问

1、如何捕获异常

了解什么是异常看这里：什么是java中的异常

捕获的方法是使用try/catch关键字。将可能产生异常，并且需要捕获的代码块使用try/catch围绕，如果产生了异常即可捕获到，将直接中断try代码块，同时执行catch代码块。

try/catch中的代码被称为受保护的代码（Protected code）。

try/catch语法：

```
try
{
//Protected code
}catch(ExceptionName e1)
{
//Catch block
}
```

如果受保护的代码发生了异常，该异常的数据类型与ExceptionName匹配，则异常会被作为catch代码块的参数传递到catch代码块中，并执行catch代码块。
例子：

```
import java.io.*;
public class ExceptionTest{

public static void main(String args[]){
try{
int a[] = new int[2];
System.out.println("Access element three :" + a[3]);//数组只有2个元素，访问第三个时会发生ArrayIndexOutOfBoundsException异常
}catch(ArrayIndexOutOfBoundsException e){
System.out.println("Exception thrown :" + e);
}
System.out.println("Out of the block");
}
}
```

这将产生以下结果:
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block

2、如何使用多个catch

一个try后面可以跟任意多个catch。

语法：

```
try
{
//Protected code
}catch(ExceptionType1 e1)
{
//Catch block
}catch(ExceptionType2 e2)
{
//Catch block
}catch(ExceptionType3 e3)
{
```

```
//Catch block
}
```

如果受保护的代码发生了异常，该异常被抛出到第一个catch块。如果异常的数据类型与ExceptionType1匹配，它就会被第一个catch捕获。如果不是，则该异常传递到第二个catch语句。以此类推，直到异常被捕获或全未匹配。若全未捕获，则终止执行，并将异常抛至调用堆栈中。

例子：

```
import java.io.*;
public class ExceptionTest{
 public static void main(String args[]){
        try {
            FileInputStream file = new FileInputStream("/usr/test");
            byte x = (byte) file.read();
        } catch (IOException i) {
            System.out.println("IOException thrown  :" + i);
        } catch (NullPointerException f) {
            System.out.println("NullPointerException thrown  :" + f);
        }
        System.out.println("catch");
}
}
```

这将产生以下结果:

IOException thrown :java.io.FileNotFoundException: /usr/test (No such file or directory)

catch

如何将异常抛出看这里：如何抛出异常

]

[

Java千百问_04异常处理（003）_如何抛出异常

,

点击进入_更多_Java千百问

1、如何抛出异常

抛出异常使用throws/throw关键字。了解异常看这里：什么是java中的异常

想知道如何捕获异常看这里：如何捕获异常

2、throws关键字是什么

throws用来声明某一个方法可能抛出的异常，这个异常可以是系统定义的，也可以是自己定义的。
调用throws修饰的方法，必须要对其做异常处理，或者将异常声明抛出（使用throws）。
语法：

```
import java.io.*;
public class className
{
public void deposit(double amount) throws RemoteException
{
// Method implementation
}
//Remainder of class definition
}
```

一个方法可以声明它抛出多个异常，在这种情况下，异常都是在以逗号分割的形式声明的。
语法：

```
import java.io.*;
public class className
{
public void withdraw(double amount) throws RemoteException,
InsufficientFundsException
{
// Method implementation
}
//Remainder of class definition
}
```

例子：

```
import java.io.*;
public class ExceptionTest{
 public static void main(String args[]) throws IOException, NullPointerException{
file = new FileInputStream(fileName);
x = (byte) file.read();
}
}
```

这将产生以下结果:

Exception in thread "main" java.io.FileNotFoundException: /usr/test (No such file or directory)

at java.io.FileInputStream.open(Native Method)
异常被抛出，中断执行，并打印了堆栈信息。

3、throw关键字是什么

使用throw关键字可以抛出一个异常对象。
另外，如果一个方法不处理异常，则该方法必须使用throws关键字声明它。

语法：

```
import java.io.*;
public class className
{
public void deposit(double amount) throws RemoteException
{
// Method implementation
throw new RemoteException();
}
//Remainder of class definition
}
```

例子：

```
public class className
{
public void main(String[] args) throws Exception
{
throw new Exception("异常");
}
}
```

这将产生以下结果:
Exception in thread "main" java.lang.Exception: 异常
at com.test.Test.main(Test.java:37)
异常被抛出，并打印了堆栈信息。

4、throws/throw关键字有什么区别

throws是用来声明一个方法可能抛出的所有异常信息。

throw则是指抛出的一个具体的异常对象。

]

[

Java千百问_04异常处理（004）_finally关键字如何使用

,

点击进入_更多_Java千百问

1、finally关键字如何使用

finally关键字用于try后面，finally块中的代码总是执行，不论是否发生异常。一般用于清理工作、关闭链接等类型的语句。了解java异常看这里：什么是java中的异常

如何捕获java中异常看这里：如何捕获异常

如何将异常抛出看这里：如何抛出异常

语法：

```
try
{
//Protected code
}catch(ExceptionType1 e1)
{
//Catch block
}catch(ExceptionType2 e2)
{
//Catch block
}catch(ExceptionType3 e3)
{
//Catch block
}finally
{
//The finally block always executes.
}
```

例子：

```
public class ExcepTest{

public static void main(String args[]){
int a[] = new int[2];
try{
System.out.println("Access element three :" + a[3]);
}catch(ArrayIndexOutOfBoundsException e){
System.out.println("Exception thrown :" + e);
}
finally{
a[0] = 6;
System.out.println("First element value: " +a[0]);
System.out.println("The finally statement is executed");
}
}
}
```

这将产生以下结果:

Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3

First element value: 6

The finally statement is executed

2、finally要注意几点

1、finally或catch语句一定会伴随try语句出现。
2、try语句不能单独使用，必须配合catch语句或finally语句。
3、try语句可以单独与catch语句一起使用，也可以单独与finally语句一起使用，当然也可以三者一起使用。
4、任何代码不能出现在try, catch, finally块之间。

]

[

Java千百问_04异常处理（005）_如何自定义异常

,

点击进入_更多_Java千百问

1、如何定义自己的异常

Java支持自己创建的异常。了解异常看这里：什么是java中的异常

方法如下：

1、所有的异常必须是Throwable的子类。
2、如果想写一个检查异常，需要扩展Exception类。
3、如果想编写一个运行时异常，则需要扩展RuntimeException类。
4、异常类与任何其他类一样，可以包含字段和方法。
我们可以定义如下自己的异常处理类：

```
class MyException extends Exception{
}
```

例子：

```
import java.io.*;

public class InsufficientFundsException extends Exception
{
private double amount;
public InsufficientFundsException(double amount)
{
this.amount = amount;
}
public double getAmount()
{
return amount;
}
}
```

为了证明我们的使用用户定义的异常，下面的CheckingAccount类包含一个withdraw()方法抛出一个InsufficientFundsException。

```
import java.io.*;

public class CheckingAccount
{
private double balance;
private int number;
public CheckingAccount(int number)
{
this.number = number;
}
public void deposit(double amount)
{
balance += amount;
}
```

```
public void withdraw(double amount) throws
InsufficientFundsException
{
if(amount <= balance)
{
balance -= amount;
}
else
{
double needs = amount - balance;
throw new InsufficientFundsException(needs);
}
}
public double getBalance()
{
return balance;
}
public int getNumber()
{
return number;
}
}
```

下面BankDemo程序演示调用deposit()和withdraw() 方法。

```
public class BankDemo
{
public static void main(String [] args)
{
CheckingAccount c = new CheckingAccount(101);
System.out.println("Depositing $500...");
c.deposit(500.00);
try
{
System.out.println("
Withdrawing $100...");
c.withdraw(100.00);
System.out.println("
Withdrawing $600...");
c.withdraw(600.00);
}catch(InsufficientFundsException e)
{
System.out.println("Sorry, but you are short $"
+ e.getAmount());
e.printStackTrace();
}
}
}
```

编译所有上述三个文件并运行BankDemo，这将产生以下结果：

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
at CheckingAccount.withdraw(CheckingAccount.java:25)
at BankDemo.main(BankDemo.java:13)
```

]

[

Java千百问_04异常处理（006）_常见的Java异常有哪些(运行时)

,

1、常见的java运行时异常有哪些

了解运行时异常看这里：什么是java中的异常
常见的非运行时异常看这里：[常见的非运行时异常有哪些][3]
[3]:

我们所说的java常见异常是指jdk或者其他常用第三方jar中，出现频次很高的异常。常见的运行时异常（RuntimeException）包括：
空指针异常类：NullPointerException
数组下标越界异常：ArrayIndexOutOfBoundsException
数组负长度异常：NegativeArraySizeException
数组存储异常：ArrayStoreException
算术异常类：ArithmeticException

非法参数异常：IllegalArgumentException
类型强制转换异常：ClassCastException
枚举常量不存在异常：EnumConstantNotPresentException
数字转换异常：NumberFormatException
无效的状态异常：IllegalStateException

无效的监控状态异常：IllegalMonitorStateException
无效的线程状态异常：IllegalThreadStateException

2、什么情况下会抛出这些异常

如何抛出异常看这里：如何抛出异常

1. **java.lang.NullPointerException**
   这个异常大家肯定都经常遇到，异常的解释是：程序遇上了空指针。简单地说就是调用了未经初始化的对象或者是不存在的对象。
   这个错误经常出现在操作方法返回值、调用数组这些操作中等等。
   一般在调用他人有可能返回null的方法时，对null进行了后续操作，会抛出该异常，这里应该首先进行null判断然后再进行后续操作。
   对数组操作中出现空指针，很多情况下是一些刚开始学习编程的朋友常犯的错误，即把数组的初始化和数组元素的初始化混淆起来了。数组的初始化是对数组分配需要 的空间，而初始化后的数组，其中的元素并没有实例化，依然是空的，所以还需要对每个元素都进行初始化（如果要调用的话）

2. **java.lang.ArrayIndexOutOfBoundsException**

这个异常相信很多朋友也经常遇到过，异常的解释是：数组下标越界（这里的数组包括各类集合，如List等）。
现在程序中大多都有对数组的操作，因此在调用数组的时候一定要认真检查，看调用的下标是不是超出了数组的范围。再通过下标获取数组值之前，最好先查看一下数组的length，以免出现这个异常。
另外，如果传入的下标为负数，也会出现这个异常。

3. **java.lang.NegativeArraySizeException**
   该异常的解释是：数组长度为负值异常。
   当使用负数作为数组长度创建数组时抛出该异常。
   在创建数组之前，一定要确保数组长度非负，尤其是隐式创建（数组长度为变量）。

4. **java.lang.ArrayStoreException**
   该异常的解释是：数组存储异常。
   当向数组中存放非数组声明类型对象时抛出。
   在为数组赋值时，一定要注意类型的一致。

5. **java.lang.ArithmeticException**
   该异常的解释是：数学运算异常。
   比如程序中出现了除以零这样的运算就会出这样的异常。对这种异常，大家只要好好检查一下自己程序中涉及到数学运算的地方，就能够解决。

6. **java.lang.IllegalArgumentException**
   该异常的解释是：非法参数异常。
   很 多java或者第三方类库中的方法在某情况下都会引发这样的错误。比如g.setcolor(int red,int green,int blue)这个方法中的三个值，如果有超过255的则会出现这个异常。一旦发现这个异常，我们要赶紧去检查一下方法调用中的参数传递是不是出现了错误。

7. **java.lang.ClassCastException**
   该异常的解释是：强制类型转换异常。
   假设有类A和B（A不是B的父类或子类），O是A的实例，那么当强制将O构造为类B的实例时抛出该异常。
   一般会在类型强制转换时出现，我们一定要梳理好继承关系采取强转即可避免。

8. **java.lang.EnumConstantNotPresentException**
   该异常的解释是：枚举常量不存在异常。
   当应用试图通过名称和枚举类型访问一个枚举对象，但该枚举对象并不包含常量时，抛出该异常。
   定义和使用枚举类型时需要小心。

9. **java.lang.NumberFormatException**
   该异常的解释是：数字转换异常。该异常继承于IllegalArgumentException。
   当将字符串转换为数字时，若格式错误则转换失败，抛出该异常。例如将”abc”抓换为整型即会抛出该异常。
   我们在做数字转换时一定要注意。

10. **java.lang.IllegalStateException**
    该异常的解释是：无效的状态异常。
    当在Java环境和应用尚未处于某个方法的合法调用状态，而调用了该方法时，抛出该异常。

例如在程序中两次调用了response.sendRedirect()方法，就会抛出该异常。

11. **java.lang.IllegalMonitorStateException**
    该异常的解释是：无效的监控状态异常。
    当某个线程试图等待一个自己并不拥有的对象（O）的监控器或者通知其他线程等待该对象（O）的监控器时，抛出该异常。
    异常的发生是由于程序员没有注意notify()，notifyAll()，wait()方法的使用条件，没有真正理解线程同步机制。如果当前的线程不是此对象锁的所有者，却调用该对象的notify()，notifyAll()，wait()方法时抛出该异常。

12. **java.lang.IllegalThreadStateException**
    该异常的解释是：无效的线程状态异常。该异常继承于IllegalArgumentException。
    当线程尚未处于某个方法的合法调用状态，而调用了该方法时，抛出异常。
    当对一个已经死亡的线程调用start，sleep之类的操作，会抛出该异常。

]

[

Java千百问_04异常处理（007）_常见的java异常有哪些(非运行时)

,

点击进入_更多_Java千百问

1、常见的java运行时异常有哪些

了解非运行时异常看这里：什么是java中的异常
常见的运行时异常看这里：常见的运行时异常有哪些

我们所说的常见异常是jdk或者其他常用第三方jar中的异常，出现频次很高的异常。常见的非运行时异常（即检查异常，checked exception）包括：

- **操作数据库异常：SQLException**
- **输入输出异常：IOException**
- **文件未找到异常：FileNotFoundException**
- **反射操作异常：ReflectiveOperationException**
- **类未找到异常: ClassNotFoundException**
- **方法未找到异常：NoSuchMethodException**
- **字段未找到异常：NoSuchFieldException**
- **非法访问权限异常：IllegalAccessException**
- **实例化异常：InstantiationException**
- **不支持克隆异常：CloneNotSupportedException**
- **被中止异常：InterruptedException**

2、什么情况下会抛出这些异常

如何抛出异常看这里：java如何抛出异常

1.java.lang.SQLException

该异常的解释是：sql异常。
sql语句异常种类十分多，通常都是sql语句、数据库执行错误导致，常见的表现有：

- invalid column name 无效列名
- table or view does not exist 表或者视图不存在
- cannot insert NULL into () 不能将空值插入
- 缺少表达式
- SQL 命令未正确结束

在操作数据库时需要考虑全面，尽量避免该异常。

2.java.lang.IOException

该异常的解释是：输入输出异常。
该异常种类也十分多（拥有很多子类），尤其对文件的操作，以及android开发。常见的表现有：

- FileNotFoundException 文件找不到。
- InvalidPropertiesFormatException 输入内容不符合属性集的正确 XML 文档类型。

3.java.lang.FileNotFoundException

该异常的解释是：文件不存在异常。该异常继承于 IOException。
这个异常通常是获取文件时，文件路径或文件名称错误导致的。

4.java.lang.ReflectiveOperationException

该异常的解释是：反射操作相关的异常。
由于反射的特殊性，类、方法、属性均使用String作为名称进行操作，对于该类异常一定要十分注意。
了解反射看这里：什么是java中的反射

5.java.lang.ClassNotFoundException

该异常的解释是：指定的类不存在。该异常继承于ReflectiveOperationException。
这个异常通常是在使用反射时，或者服务端引入jar包时抛出。
使用反射时，根据类名（字符串）获取Class时，包、类名有误会造成该异常。

6.java.lang.NoSuchMethodException

该异常的解释是：指定的方法不存在。该异常继承于ReflectiveOperationException。
这个异常通常是在使用反射时抛出。
使用反射时，根据方法名（字符串）调用Method时，方法名有误会造成该异常。

7.java.lang.NoSuchFieldException

该异常的解释是：指定的字段不存在。该异常继承于ReflectiveOperationException。
这个异常通常是在使用反射时抛出。
使用反射时，根据字段名（字符串）获取、操作Field时，字段名有误会造成该异常。

8.java.lang.IllegalAccessException

该异常的解释是：没有访问权限。
当应用程序要调用一个类，但当前的方法即没有对该类的访问权限便会出现这个异常。
最常见的地方即在使用反射调用private方法/属性时会抛出该异常，将private方法/属性共有化public即可。
想了解public和privite看这里：public、privite、protected有什么区别

9.java.lang.InstantiationException

该异常的解释是：实例化异常。该异常继承于ReflectiveOperationException。
当试图通过newInstance()方法创建某个类的实例，而该类是一个抽象类或接口时，抛出该异常。

10.java.lang.CloneNotSupportedException

该异常的解释是：不支持克隆异常。该异常继承于 ReflectiveOperationException。
当没有实现Cloneable接口或者不支持克隆方法时,调用其clone()方法则抛出该异常。

11.java.lang.InterruptedException

该异常的解释是：被中止异常。
当某个线程处于长时间的等待、休眠或其他暂停状态，而此时其他的线程通过Thread的interrupt方法终止该线程时抛出该异常。

]

[

Java千百问_04异常处理（008）_java中常见的错误有哪些

,

点击进入_更多_Java千百问

java中常见的错误有哪些

想了解异常看这里：什么是java中的异常
常见的运行时异常看这里：常见的Java异常有哪些(运行时)
常见的非运行时异常看这里：常见的java异常有哪些(非运行时)
java中除了异常Exception之外，还有一大类错误，即ERROR，我们常见的ERROR如下：

1、java.lang.Error

错误。是所有错误的基类，用于标识严重的程序运行问题。这些问题通常描述一些不应被应用程序捕获的反常情况。
原因:

1. 对系统所访问外部资源，未执行关闭操作，导致外部资源大量浪费，最终可-能导致系统无法正常运行；
2. 对系统所访问的外部资源关闭次数太多，外部系统无法正常处理；
3. 系统访问的外部资源出现异常情况。

解决方案：

1. 访问外部资源前，首先检查该资源(如数据库)是否可正常连接或操作。
2. 访问外部资源时，如果进行了连接，一定进行关闭操作，并仅进行一次关闭操作。
3. 尽量在同一操作中共享外部资源，以减少该操作对资源的消费，提高程序的执行效率。

2、java.lang.AbstractMethodError

抽象方法错误。当应用试图调用抽象方法时抛出。

3、java.lang.AssertionError

断言错。用来指示一个断言失败的情况。

4、java.lang.ClassCircularityError

类循环依赖错误。在初始化一个类时，若检测到类之间循环依赖则抛出该异常。

5、java.lang.ClassFormatError

类格式错误。当Java虚拟机试图从一个文件中读取Java类，而检测到该文件的内容不符合类的有效格式时抛出。

6、java.lang.ExceptionInInitializerError

初始化程序错误。当执行一个类的静态初始化程序的过程中，发生了异常时抛出。静态初始化程序是指直接包含于类中的static语句段。

7、java.lang.IllegalAccessError

违法访问错误。当一个应用试图访问、修改某个类的域（Field）或者调用其方法，但是又违反域或方法的可见性声明，则抛出该异常。

8、java.lang.IncompatibleClassChangeError

不兼容的类变化错误。当正在执行的方法所依赖的类定义发生了不兼容的改变时，抛出该异常。一般在修改了应用中的某些类的声明定义而没有对整个应用重新编译而直接运行的情况下，容易引发该错误。

9、java.lang.InstantiationError

实例化错误。当一个应用试图通过Java的new操作符构造一个抽象类或者接口时抛出该异常.

10、java.lang.InternalError

内部错误。用于指示Java虚拟机发生了内部错误。

11、java.lang.LinkageError

链接错误。该错误及其所有子类指示某个类依赖于另外一些类，在该类编译之后，被依赖的类改变了其类定义而没有重新编译所有的类，进而引发错误的情况。

12、java.lang.NoClassDefFoundError

未找到类定义错误。当Java虚拟机或者类装载器试图实例化某个类，而找不到该类的定义时抛出该错误。

13、java.lang.NoSuchFieldError

域不存在错误。当应用试图访问或者修改某类的某个域，而该类的定义中没有该域的定义时抛出该错误。

14、java.lang.NoSuchMethodError

方法不存在错误。当应用试图调用某类的某个方法，而该类的定义中没有该方法的定义时抛出该错误。

15、java.lang.OutOfMemoryError

内存不足错误。当可用内存不足以让Java虚拟机分配给一个对象时抛出该错误。

16、java.lang.StackOverflowError

堆栈溢出错误。当一个应用递归调用的层次太深而导致堆栈溢出时抛出该错误。

17、java.lang.ThreadDeath

线程结束。当调用Thread类的stop方法时抛出该错误，用于指示线程结束。

18、java.lang.UnknownError

未知错误。用于指示Java虚拟机发生了未知严重错误的情况。

19、java.lang.UnsatisfiedLinkError

未满足的链接错误。当Java虚拟机未找到某个类的声明为native方法的本机语言定义时抛出。

20、java.lang.UnsupportedClassVersionError

不支持的类版本错误。当Java虚拟机试图从读取某个类文件，但是发现该文件的主、次版本号不被当前Java虚拟机支持的时候，抛出该错误。

21、java.lang.VerifyError

验证错误。当验证器检测到某个类文件中存在内部不兼容或者安全问题时抛出该错误。

22、java.lang.VirtualMachineError

虚拟机错误。用于指示虚拟机被破坏或者继续执行操作所需的资源不足的情况。

]