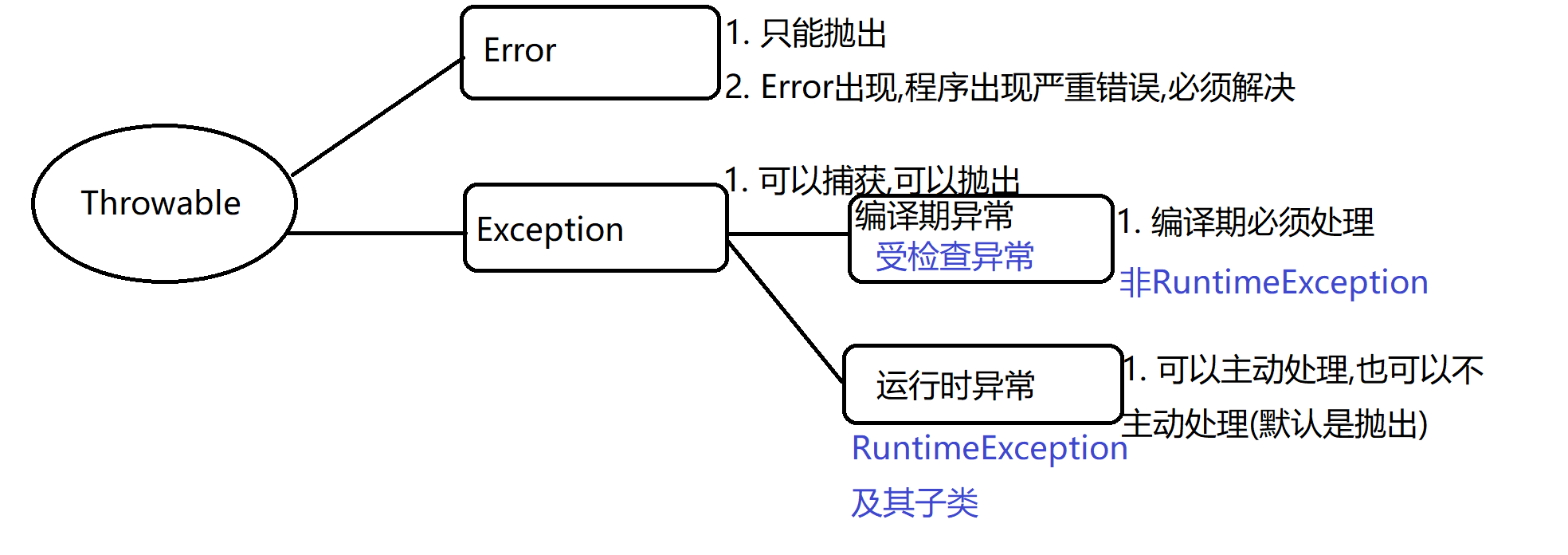
# ---+-/\*异常

异常就是程序运行过程中不正常执行.

# 异常体系



1 Throwable是所有异常(Exception)和错误(Error)的超类.

2 Error是严重错误,不能捕获,只能抛出!

3 Exception异常.可以捕获,可以抛出

4 Exception异常又可分为

编译期异常(写完代码,编译即出错),又叫受检查异常.非RuntimeException及其子类都是编译期异常.必须处理.

运行时异常(写完正常,运行时会报错),又叫不受检查异常.只有RuntimeException及其子类是运行时异常.可以处理可以不出处理.

# 异常的处理

## 抛出(throws)

语法:

1方法的签名/声明,即参数列表后大括号前使用throw**s**声明要抛出的异常**类名**.

2 throws后可以跟多个异常类名,之间使用逗号,隔开

3 程序一旦抛出异常,在方法抛出处,后续代码不再执行

ps: 方法声明抛出异常,并不是一定会抛出异常.只是有可能抛出异常.

|  |
| --- |
| **package** com.zhiyou100;  **import** javax.lang.model.type.ArrayType; **import** java.lang.reflect.Member; **import** java.text.ParseException; **import** java.text.SimpleDateFormat; **import** java.util.Date; */\*\*  \* alt+enter 快捷键  \* 代码提示  \* 导包  \* 补全本地变量  \* 创建没有的方法  \* 修改类名*  *\* -----*  *\* alt+ctrl+上下,快速上下复制*  *\* alt+上下. 快速上下移动*  *\* ctrl+d 删除一行  \*/ /\*\*  \** ***@author*** *QiuShiju  \** ***@date*** *2021/1/4  \** ***@desc*** *\*/* **public class** Demo2 {    **public static void** main(String[] args) **throws** ParseException{  */\*\*  \* 抛出异常,是将异常向上抛给方法的调用者  \* 如果是主方法,那么异常就抛给虚拟机  \* 虚拟机直接输出到控制台  \*/* System.***out***.println(**"main - 1"** );  *m3*();*// ArithmeticException   // 调用有抛出编译期异常的方法时,也必须强制处理  // m4();// ParseException* System.***out***.println(**"main - 2"** );  }   *// 方法有编译期异常 // private static void m4() throws ParseException{ // }* **private static void** m3() **throws** ArithmeticException{  System.***out***.println(**"m3 - 1"** );  System.***out***.println(2/0 );  System.***out***.println(**"m3 - 2"** );  }    *// shift + alt + m 抽取方法* **private static void** m2() {  */\*\*  \* 程序一旦抛出异常,在方法抛出处,后续代码不再执行  \*/* System.***out***.println(1 );  System.***out***.println(2/0 );  System.***out***.println(3 );  }   */\*\*  \* 异常处理方式一: 抛出异常  \*/* **public static void** m1() **throws** ParseException,ArithmeticException{  *// 编译期异常,抛出* Date date = **new** SimpleDateFormat(**"yyyy-MM-dd"**).parse(**"2021年01月04日"**);  System.***out***.println(date );   *// 运行时异常,可以抛出,也可以不主动处理.  // 如果不主动处理,那么虚拟机默认就是将异常抛出* System.***out***.println(1/0 );  } } |

## 试图捕获(try catch)

语法:

try{

// 要执行的代码,可能出现异常的代码

}catch(异常类 对象名) {

// 捕获到异常后,执行的动作

}

1 只有try中有异常出现,catch才会执行

2 在try代码块内部如果有异常,在try中抛出异常处,不再向下执行;

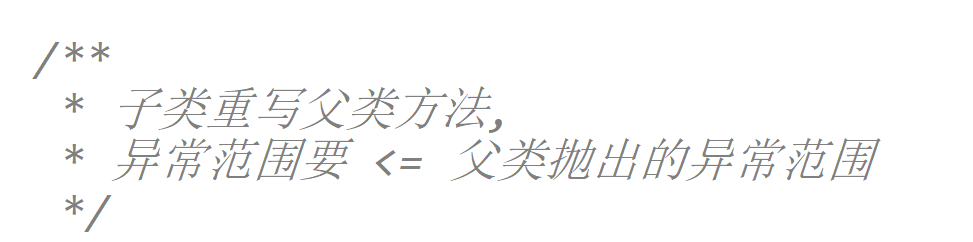
但是整个try,catch结构的后面,无论是否有异常,都会正常执行

3 可以同时写多个catch捕获异常

4 catch后声明捕获的,要与try中抛出的异常一致,才可以捕获成功,否则继续抛出

|  |
| --- |
| **package** com.zhiyou100;  **import** java.text.ParseException; **import** java.text.SimpleDateFormat; **import** java.util.Date;  */\*\*  \** ***@author*** *QiuShiju  \** ***@date*** *2021/1/4  \** ***@desc*** *异常处理方法二: 捕获异常  \*/* **public class** Demo3 {   **public static void** main(String[] args) {  *m5*( );   }   */\*\*  \* 演示: catch捕获异常与try中抛出异常是否一致问题  \*/* **private static void** m5() {  System.***out***.println(1 );  **try**{  System.***out***.println(2 );  System.***out***.println(1/0 );*// 抛出算术异常* System.***out***.println(3 );  */\*\*  \* 捕获的异常类型要和抛出的一致,  \* 否则捕获不到.  \* 如果捕获不到,则继续抛出异常.  \*/* }**catch** (RuntimeException ie) { *// 捕获下标越界异常* System.***out***.println(4 );   }  System.***out***.println(5 );  }   */\*\*  \* 演示: 可以同时写多个catch捕获异常  \*/* **private static void** m4() {  **try**{   **int**[] arr = {11,22,33};  System.***out***.println(arr[2] );  System.***out***.println(1/0 );    */\*\*  \* catch (Exception ae) { // 捕获下标越界异常   }  如果写同时写多个catch捕获异常,排在前面的异常范围不能比下面的大  ---> 关于重写时方法异常问题?  子类重写父类方法,子类的方法的异常范围不能比父类异常范围大.  绝对的,父类方法没有抛出异常,那么子类也不能抛出异常  \*/* }**catch** (ArrayIndexOutOfBoundsException re) { *// 捕获下标越界异常* }**catch** (ArithmeticException ae) { *// 捕获算术异常* System.***out***.println(**"算术异常!"** );  }  }   */\*\*  \* 捕获异常结构,代码执行流程  \*/* **private static void** m3() {  System.***out***.println(1 );  **try** {  System.***out***.println(11 );  System.***out***.println(1/0 );  *// try中抛出异常的后面不再执行* System.***out***.println(12 );*// 不再执行* }**catch** (ArithmeticException ae) {  System.***out***.println(2 );  }  *// trycatch后续的代码,无论是否有异常,都会执行* System.***out***.println(3 );  }   **private static void** m2() {  **try**{  System.***out***.println(1/0 );  }**catch** (ArithmeticException ae) {  System.***out***.println(**"算术异常!"** );  }  }   */\*\*  \* 演示捕获异常  \*/* **private static void** m1() {  **try**{  Date date =  **new** SimpleDateFormat(**"yyyy-MM-dd"**).  parse(**"2021/01/04"**);  System.***out***.println(date );  }**catch** (ParseException pe) {  */\*\*  \* 抓到异常后,可以记录异常到日志文件,或者数据库等位置  \* 便于后期查找bug,维护项目  \*/* System.***out***.println(**"抓到异常!!"** );  }  } } |

## ps: 重写方法时异常问题



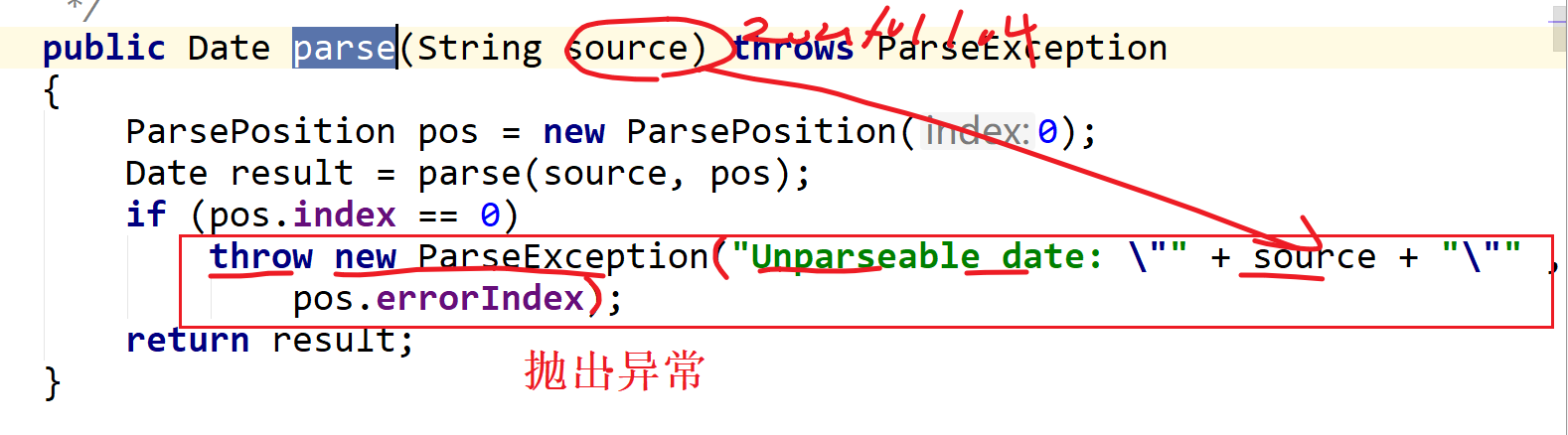
# 如何手动抛出异常

在上述演示过程中,所有异常都是程序运行时,虚拟机抛出异常.

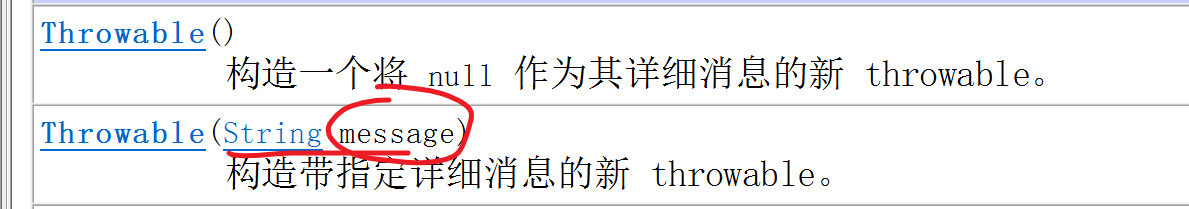
那么我们自己如何主动抛出异常?

## throw

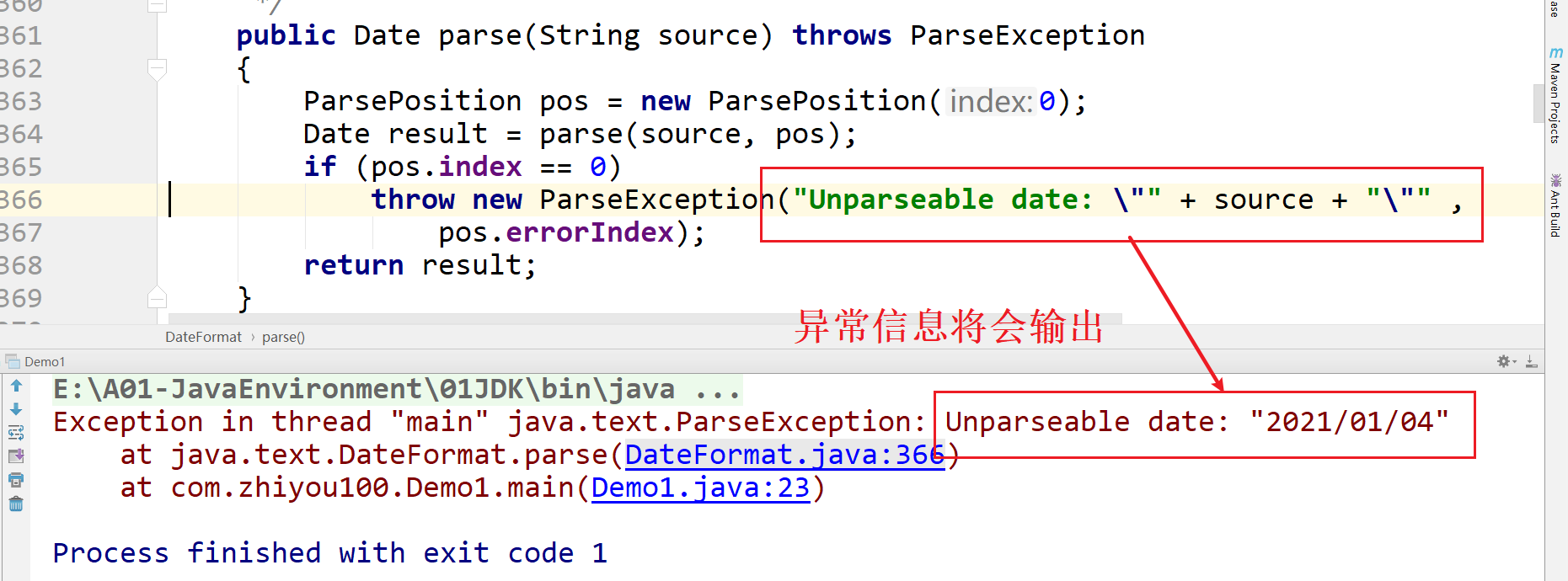
在需要的地方,直接使用throw关键字 + 异常对象即可将异常抛出.



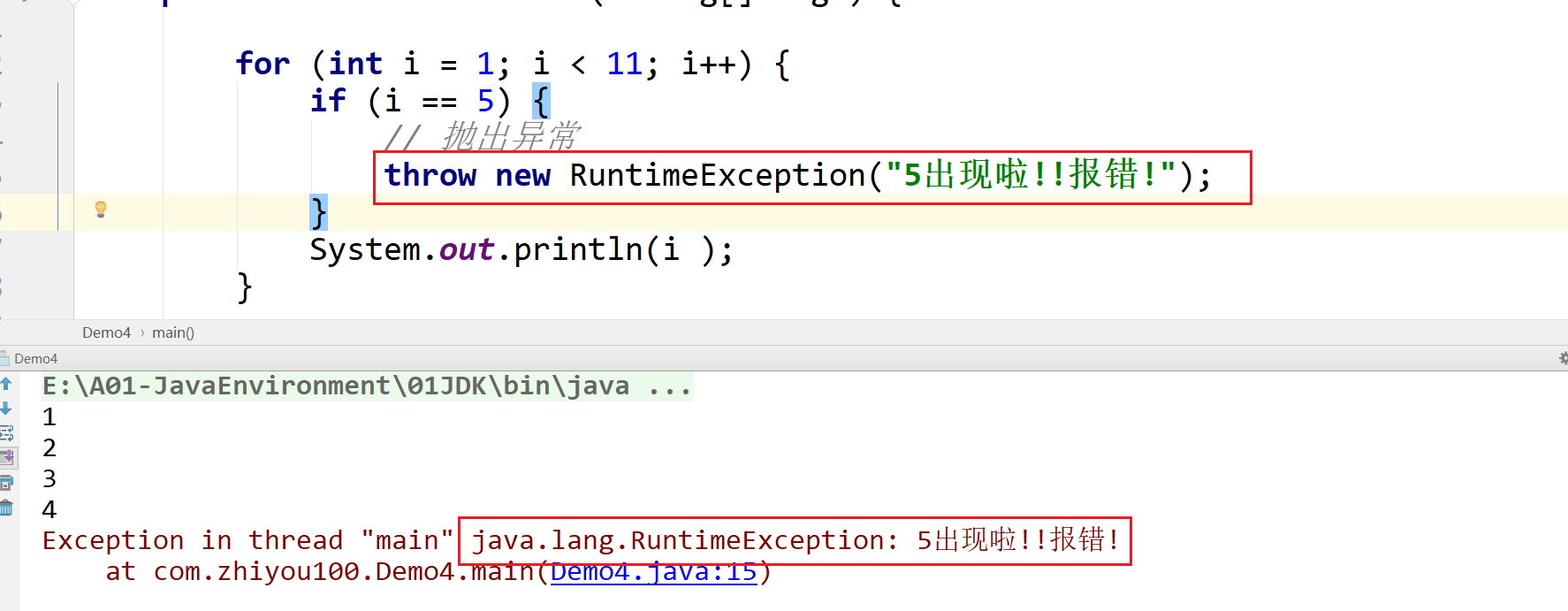
构造方法中传入的是异常信息



作用:在输出异常信息时,会输出



## 自行演示



throw 和 throws 的区别

1 throw后面跟异常对象,throws异常类名

2 throw后面跟一个对象,throws后面跟多个类名

3 throw在方法内部使用,throws 在方法签名上使用

# 自定义异常类

只有当对象是此类Throwable（或其子类之一）的实例时，才能通过 Java 虚拟机或者 Java throw 语句抛出。

练习: 设计方法,传入人名字符串.

如果名字长度是4以上(不包含4),就抛出NameTooLongException

如果名字长度是2以下,就抛出NameTooShortException

其他即提示,人名正常

提示:

自定义异常类NameTooLongException, NameTooShortException

1 新建自定义异常类

2 继承Throwable或其子类

ps: 继承自哪个类,就拥有哪种异常的特点

3 创建有参构造,将异常信息传递给父类

4 在需要使用的地方,使用throw将异常抛出

|  |
| --- |
| **package** com.zhiyou100;  */\*\*  \** ***@author*** *QiuShiju  \** ***@date*** *2021/1/4  \** ***@desc*** *\*/* **public class** TestException {   **public static void** main(String[] args) **throws** Exception{  *checkName*(**"刘"**);  }   **public static void** checkName(String name) **throws** NameTooLongException, NameTooShortException {  **if** (name.length( ) > 4) {  *// 抛出名字太长异常* **throw new** NameTooLongException(**"名字太短长:\""**+name+**"\""**);   } **else if** (name.length( ) < 2) {  *// 抛出名字太短异常* **throw new** NameTooShortException(**"名字太短:\""**+name+**"\""** );   }  System.***out***.println(**"名字刚好,可以注册!"** );  }  } |
| **package** com.zhiyou100;  */\*\*  \** ***@author*** *QiuShiju  \** ***@date*** *2021/1/4  \** ***@desc*** *\*/* **public class** NameTooShortException **extends** Exception {   **public** NameTooShortException(String message) {  *// 将异常信息,向上传递给父类* **super**(message);  } } |
| **package** com.zhiyou100;  */\*\*  \** ***@author*** *QiuShiju  \** ***@date*** *2021/1/4  \** ***@desc*** *名字太长异常  \*/* **public class** NameTooLongException **extends** Exception{  **public** NameTooLongException(String message) {  **super**(message);  } } |

# 异常的方法

|  |  |
| --- | --- |
| [String](mk:@MSITStore:E:\\000_J_A_V_A\\ZhiyouWorkSpace\\讲课\\00%20总\\邱世举_Java\\API\\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html" \o "java.lang 中的类) | **[getMessage](mk:@MSITStore:E:\\000_J_A_V_A\\ZhiyouWorkSpace\\讲课\\00%20总\\邱世举_Java\\API\\JDK_API_1_6_zh_CN.CHM::/java/lang/Throwable.html" \l "getMessage())**()            返回此 throwable 的详细消息字符串。 |
| [StackTraceElement](mk:@MSITStore:E:\\000_J_A_V_A\\ZhiyouWorkSpace\\讲课\\00%20总\\邱世举_Java\\API\\JDK_API_1_6_zh_CN.CHM::/java/lang/StackTraceElement.html" \o "java.lang 中的类)[] | **[getStackTrace](mk:@MSITStore:E:\\000_J_A_V_A\\ZhiyouWorkSpace\\讲课\\00%20总\\邱世举_Java\\API\\JDK_API_1_6_zh_CN.CHM::/java/lang/Throwable.html" \l "getStackTrace())()            提供编程访问由 输出的堆栈跟踪信息。** |
| void | **[printStackTrace](mk:@MSITStore:E:\\000_J_A_V_A\\ZhiyouWorkSpace\\讲课\\00%20总\\邱世举_Java\\API\\JDK_API_1_6_zh_CN.CHM::/java/lang/Throwable.html" \l "printStackTrace())()            将此 throwable 及其追踪输出至标准错误流。** |
| [String](mk:@MSITStore:E:\\000_J_A_V_A\\ZhiyouWorkSpace\\讲课\\00%20总\\邱世举_Java\\API\\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html" \o "java.lang 中的类) | **[toString](mk:@MSITStore:E:\\000_J_A_V_A\\ZhiyouWorkSpace\\讲课\\00%20总\\邱世举_Java\\API\\JDK_API_1_6_zh_CN.CHM::/java/lang/Throwable.html" \l "toString())()            返回此 throwable 的简短描述** |

|  |
| --- |
| **package** com.zhiyou100;  **import** java.util.Arrays;  */\*\*  \** ***@author*** *QiuShiju  \** ***@date*** *2021/1/4  \** ***@desc*** *演示异常常用方法  \*/* **public class** Demo5 {   **public static void** main(String[] args) {  System.***out***.println(1 );  **try** {  System.***out***.println(1/0 );  }**catch** (ArithmeticException ae) {  *// getMessage(); 获得异常信息.即创建异常时,构造方法中的参数* String message = ae.getMessage( );  *//System.out.println(message );   // printStackTrace(); 打印堆栈信息即完整异常现象  //ae.printStackTrace();   // toString(); 以字符串的形式返回异常信息,没有异常位置  // System.out.println(ae.toString() );   // StackTraceElement[] getStackTrace();* StackTraceElement[] stackTrace = ae.getStackTrace( );  *// System.out.println(Arrays.toString(stackTrace ) );* **for** (**int** i = 0; i < stackTrace.**length**; i++) {  System.***out***.println(**"抛异常的类: "**+stackTrace[i].getClassName() );  System.***out***.println(**"抛异常的方法: "**+stackTrace[i].getMethodName() );  System.***out***.println(**"抛异常的行号: "**+stackTrace[i].getLineNumber() );  System.***out***.println(**"抛异常的原因: "**+message );  }  }  System.***out***.println(2 );  }  } |

# 总结

1 异常是什么

2 异常的分类,各自特点

3 异常的处理方法

4 自定义异常

5 异常api

# 作业

学生类,属性姓名,年龄.

赋值时如果名字长度是4以上(不包含4),就抛出编译期异常NameTooLongException

如果名字长度是2以下,就抛出编译期异常NameTooShortException

其他正常赋值

--

如果年龄大于130,抛出运行时异常AgeGT130Exception

如果年龄小于0,就抛出运行时异常AgeLT0Exception

其他正常赋值