# 集合类的泛型

List Set Queue Map

List list = new ArrayList();

list.add("字符串");

Object str = list.get(0);

String str2 = (String) str;

可以利用泛型，来解决以上提到的强转问题：

List<泛型类型> list = new ArrayList<泛型类型>();

如果已知集合中要存储String类型，就将泛型类型替换为String;

如果已知集合中要存储Student类型，就将泛型类型替换为Student;

...

泛型类型不能是基本类型， int,double ...

但可以是包装类型： Integer, Double ...

在进行变量赋值时，两边的泛型声明也需一致

list泛型：

// 创建泛型集合 jdk 6.0

List<String> list = new ArrayList<String>();

// jdk 7.0

List<String> list2 = new ArrayList<>();

list.add("hello");

// list.add(1); // 会在编译阶段检查出错误,避免把其他类型放入集合

// 可以直接获取字符串，不需要强制转换了

String str = list.get(0);

System.out.println(str);

// 遍历集合时

for(String s : list){

System.out.println(s + " 长度:" + s.length());

}

list.remove(0);

map 泛型

// key的泛型 值的泛型

Map<String, String> map = new HashMap<>();

Map<String, String> map2 = new HashMap<String,String>();

map.put("bj", "北京");

Map<Integer, String> map3 = new HashMap<>();

map3.put(1, "北京");

map3.put(2, "天津");

map3.put(3, "河北");

map3.put(4, "山东");

// 遍历集合

System.out.println("=======================");

for(Integer key :map3.keySet()) {

System.out.println("key:" + key + " value:" + map3.get(key));

}

System.out.println("=======================");

for(Entry<Integer,String> e :map3.entrySet()){

System.out.println(e.getKey() + " " + e.getValue());

}

泛型擦除： 泛型的信息只在源码级别生效，源码被编译为\*.class以后，泛型信息就会被擦除掉，不会保留

# 异常处理

java中异常： 例如输入的数据不在处理范围内

读取文件时，文件损坏了

网络发生了中断

java中设计了用来表示这些出错情况的java类，这些类称为异常类

分了三大类

1. checked Exception 检查异常

本意是指意料之内的（程序员在设计代码时考虑到的）异常

共同父类: java.lang.Exception

1. Error 错误 （这时候代码不会继续运行了，人已经无法控制了）

网络中断

类加载错误

共同父类： java.lang.Error

1. Runtime Exception 运行时异常

本意指意料之外的异常 （算数运算中 除以0）

共同父类： java.lang.RuntimeException

## 处理异常的语法：

try {// 尝试

// 代码正常

// 就会执行完try 内所有代码

} catch () { // 捕获，捉住

// 一旦try块内的某行代码出现异常，程序就会进入catch块的代码

// 捕获异常进行处理

} finally {

// 最终都会被执行

}

try 和 finally 块只能有一个， catch块可以有多个：

try {

}

catch(异常类型1 e) { ... }

catch(异常类型2 e) { ... }

...

catch(异常类型n e) { ... }

finally {

}

例：

// 选中这些行，按tab

// 要向前移动这些行，按住shift+tab

try {

int i = 10;

int j = 0;

int k = i / j ;

// 当上一行代码出现异常，下面的所有代码不会继续运行， 属于运行时异常

System.out.println("结果:" + k);

}

catch(ArithmeticException e) { // 用来处理异常, () 声明要处理的异常类型

// e 代表一个异常对象，存储了发生了异常时的错误信息

// catch 让程序恢复运行

System.out.println("进入catch快继续运行");

System.out.println("错误信息：" + e.getMessage());

System.out.println("除数不能为0");

System.out.println("=======================");

// 打印详细的错误栈信息: 行号，那个方法执行时出现的错误

e.printStackTrace();

String str = null;

str.length();

}

finally {

/\* 无论try块内的代码执行了，还是出现了异常

\* finally 块的代码一定会被执行

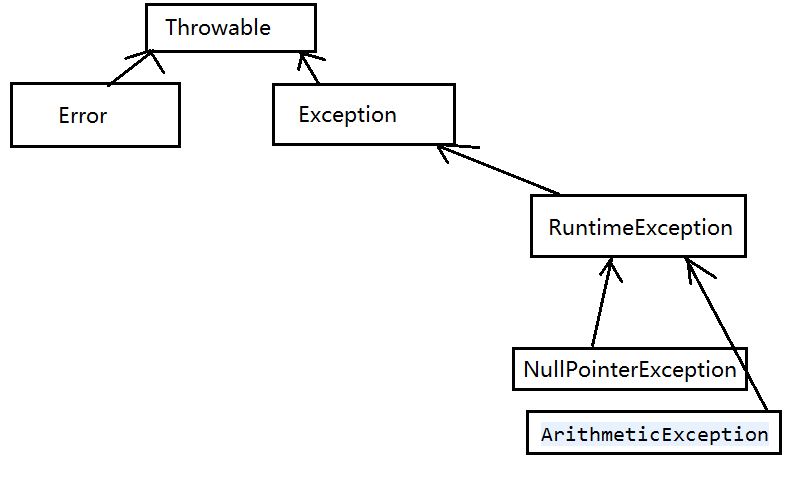
\*/

System.out.println("finally代码");

}

System.out.println("别的代码");

## 异常之间的关系：



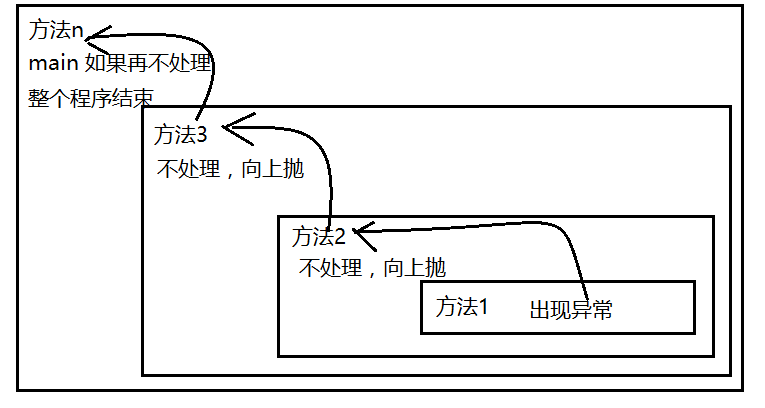
|  |  |  |  |
| --- | --- | --- | --- |
| 异常类型 | 是否需要强制加try-catch捕获 | 是否要强制加Throws 关键字 | 常见异常类型 |
| RuntimeException(运行时异常) | 否 | 否 | NullPointerException  ArithmeticException |
| Exception  (检查异常) | 是 | 是 | IOException  FileNotFoundException |
| Error(错误) | 否 | 否 | ExceptionInInitializerError |

## 处理异常的另一个选择

在某一个方法，希望不是由本方法来处理异常，交由其他方法来处理

throws 异常类型 ： 位于方法的声明部分，用来指明方法可能会抛出哪些异常

throw 异常对象 ： 作用是将异常重新抛出



## 自定义异常

自己创建异常类型

Error RuntimeException Exception

其实就是以上三种异常类型选择其一，创建子类

public class InputException extends RuntimeException {

// 无参构造

public InputException() {

super();

}

// message 代表错误信息， cause 代表异常起因

public InputException(String message, Throwable cause) {

super(message, cause);

}

// message 代表错误信息

public InputException(String message) {

super(message);

}

// cause 异常的起因（你的异常是由其他异常引起）

public InputException(Throwable cause) {

super(cause);

}

}

使用 throw new 异常对象（错误信息）; 触发此异常

## 方法覆盖时的异常处理

public class Super {

public void test1() throws Exceptin {

}

}

public class Sub extends Super{

public void test1() throws FileNotFoundException {

}

}

// 子类重写方法抛出的异常，不能比父类抛出的异常更宽

// 子类重写方法可以不抛异常

# File

File 类型可以代表磁盘文件、或磁盘目录（文件夹）

File file = new File("d:/2.txt"); //代表一个文件

File file = new File("d:/"); // 代表d盘根目录

C:\\Users\\adminstator\\Desktop\\java基础\\day01\\01\_java环境.avi

File.exists() 判断文件或目录是否已经存在

File.mkdir() 如果目录不存在，则创建目录

File.mkdirs() 一次创建多级目录

File.isDirectory() 用来判断是否是一个目录

如果是目录 File.listFiles() 列出子目录以及文件

File.delete() 删除文件，或文件夹

例子：

File file = new File("d:/a/b/c/d");

File file2 = new File("d:\\2.txt"); // 绝对路径

System.out.println("文件是否已存在:" + file.exists());

// file.mkdir(); // 创建目录

file.mkdirs(); // 创建多级目录

// 相对路径的问题

File file3 = new File("1.txt");

// 找的是程序运行时所在文件夹

System.out.println("绝对路径是：" + file3.getAbsolutePath());

System.out.println(file3.exists());

// isDirectory用来判断是否是一个目录

System.out.println("是否是目录?" + file3.isDirectory());

System.out.println("是否是目录?" + file.isDirectory());

System.out.println("================");

File file4 = new File("d:/a");

File[] all = file4.listFiles();

for(File fs: all) {

System.out.println(fs + " 是否是目录?" + fs.isDirectory());

if(fs.isDirectory()) {

fs.listFiles();

}

}

System.out.println("删除文件");

File file5 = new File("D:\\a\\b\\c\\d\\e1.txt");

file5.delete();

// 尝试删除还有内容的文件夹

File file6 = new File("d:\\a\\b\\c\\d");

file6.delete();

//返回值true，表示成功删除，返回值是false 表示删除失败

// 重命名 (移动)

// File file7 = new File("d:/2.txt");

// file7.renameTo(new File("d:/1.txt"));

File file8 = new File("d:/1.txt");

System.out.println("文件长度:" + file8.length());

## RandomAccessFile

// 参数1： 文件路径

// 参数2：使用文件的方式

// "r" 只读 "w" 只写 "rw" 可以同时读写

RandomAccessFile raf = new RandomAccessFile("e:/1.txt", "rw");

// write 写

// raf.writeInt(10);

// raf.writeInt(1);

// read 读

// System.out.println(raf.readInt());

// System.out.println(raf.readInt());

// System.out.println(raf.readByte());

// System.out.println(raf.readByte());

// System.out.println(raf.readByte());

// System.out.println(raf.readByte());

// 移动指针至某一位置, 参数表示要移动的字节数

raf.seek(3);

System.out.println(raf.readByte());

// 用完要关闭

raf.close();

补充：

日期和字符串之间的转换

java.util.Date java.lang.String