# 异常

2018年7月1日 14:58

```
/*
 * 异常：程序出现了不正常的情况
 * 程序的异常：Throwable
 *                 严重问题:Error 不处理（问题很严重，比如内存溢出）
 *                 问题：Exception
 *                         编译期问题:不是RuntimeException的异常 必须进行处理，否则编译不能通过
 *                         运行期问题:RuntimeException 不处理，代码不够严谨，需要修正代码
 * 如果程序出现问题，我们没有处理，jvm默认处理
 * 把异常的名称、原因及出现的问题等信息输出在控制台，同时会结束程序
 */
public class ExceptionDemo {
	public static void main(String[] args) {
		int a = 10;
		// int b=2;
		int b = 0;
		System.out.println(a / b);//ArithmeticException--->RuntimeException
		System.out.println("over");
	}
}
```

# 异常处理

2018年7月18日 23:17

```
/*
 * 如果程序出现问题，我们没有处理，jvm默认处理
 * 自己处理：
 *                 1.try...catch...finally
 *                 格式：
 *                         try{
 *                                 可能出现的代码
 *                         }catch(异常名变量){
 *                                 针对问题的处理
 *                         }finally{
 *                                 释放资源
 *                         }
 *                 变形格式
 *                         try{
 *                                 可能出现的代码
 *                         }catch(异常名变量){
 *                                 针对问题的处理
 *                         }
 *                 try里面的代码越少越好,catch里必须有内容，哪怕是一个简单的提示
 *                 2.throws 抛出
 */
public class ExceptionDemo {
        public static void main(String[] args) {
                int a = 10;
                // int b=2;
                int b = 0;

                try {
                        System.out.println(a / b);
                }catch(ArithmeticException ae){
                        System.out.println("除数不能为0");
                }
                System.out.println("over");
        }
}
```

# 多个异常处理

2018年7月18日 23:42

```
/*
 * A.一个异常
 * B.两个异常
 *                 1.每一个写一个try...catch()
 *                 2.写一个try,多个catch
 *                 try{}
 *                 catch(异常类名,变量名){}
 *                 catch(异常类名,变量名){}
 *                 ......
 *                 注意：能明确的尽量明确，不要不要用大的处理，效率低
 *                 Exception必须放最后，平级关系顺序无所谓，父必须在子后面
 */
public class ExceptionDemo2 {
	public static void main(String[] args) {
		method1();
		method2();// 除数不能为0 over 只走一个异常
		method3();
	}

	public static void method1() {
		int a = 10;
		int b = 0;
		try {
			System.out.println(a / b);
		} catch (ArithmeticException e) {
			System.out.println("除数不能为0");
		}

		int[] arr = { 1, 2, 3 };
		try {
			System.out.println(arr[3]);
		} catch (ArrayIndexOutOfBoundsException e) {
			System.out.println("索引越界");
		}
		System.out.println("over");
	}
```

```
    public static void method2() {
        int a = 10;
        int b = 0;
        int[] arr = { 1, 2, 3 };
        try {
            System.out.println(a / b);
            System.out.println(arr[3]);
        } catch (ArithmeticException e) {
            System.out.println("除数不能为0");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("索引越界");
        }
        System.out.println("over");
    }

    public static void method3() {
        int a = 10;
        int b = 0;
        int[] arr = { 1, 2, 3 };
        try {
            System.out.println(a / b);
            System.out.println(arr[3]);
            System.out.println("出现了一个不清楚的异常");
        } catch (ArithmeticException e) {
            System.out.println("除数不能为0");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("索引越界");
        } catch (Exception e) {
            System.out.println("出问题了");
        }
        System.out.println("over");
    }
}
```

# JDK7新特性

2018年7月19日 10:33

```
/*
 * JDK7出现了新的异常处理方案：
 *                  try{
 *                  }catch(异常名1|异常名2|异常名2 ... 变量){
 *                          ......
 *                  }
 *                  缺点：  1.处理方式是一致的
 *                                  2.多个异常间必须是平级关系
 */
public class ExceptionDemo3 {
        public static void main(String[] args) {
                method();
        }

        public static void method() {
                int a = 10;
                int b = 0;
                int[] arr = { 1, 2, 3 };
                try {
                        System.out.println(a / b);
                        System.out.println(arr[3]);

                } catch (ArithmeticException | ArrayIndexOutOfBoundsException e) {
                        System.out.println("出问题了");
                }
                System.out.println("over");
        }
}
```

# 编译期异常

2018年7月19日 10:50

```
/*
 * 编译时异常与运行时异常的区别：
 * 编译期异常：Java程序必须显示处理，否则程序就会发生错误，无法通过编译
 * 运行期异常：无序显示处理，也可以和编译时异常一样处理
 */
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/*
 *
 */
public class ExceptionDemo {
	public static void main(String[] args) {
		int a = 0;
		int b = 0;
		// System.out.println(a/b);//运行期异常，通过增强逻辑修改

		String s = "2018-07-19";
		SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
		try {
			Date d = sdf.parse(s);
			System.out.println(d);
		} catch (ParseException e) {
			// e.printStackTrace();
			System.out.println("解析日期出错");
		}
	}
}
```

# 异常的方法

2018年7月19日 11:13

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/*
 * 在try里面发生问题后，jvm会生成一个异常对象，然后把这个对象抛出，和catch里面的类进行匹配
 * 如果该对象是某个类型的，就会执行该catch里面的处理信息
 * 异常中的方法：
 *              public String getMessage():异常的消息字符串
 *              public String toString():返回异常的简单信息描述
 *                      此对象的类的name（全路径名）
 *                      ": "冒号和一个空格
 *                      调用此对象getLocalizedMessage()方法的结果（getMessage()内容）
 *              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
 */
public class ExceptionDemo {
    public static void main(String[] args) {

        String s = "2018-07-19";
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        try {
            Date d = sdf.parse(s);
            System.out.println(d);
        } catch (ParseException e) {
            // e.printStackTrace();

            // getMessage
            System.out.println(e.getMessage());// Unparseable date: "2018-07-19"

            // toString()
            System.out.println(e.toString());
            // java.text.ParseException: Unparseable date: "2018-07-19"
```

```
                e.printStackTrace();
            }
            System.out.println("over");//可以输出
        }
}
```

# throws方法

2018年7月19日 11:59

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/*
 * 有些时候，我们没有权限去处理某个异常
 * 为了解决出错问题，java提供了抛出的处理方案
 * 格式:
 *                  throws 异常类名
 *                  注意：这个格式必须跟在方法的括号后
 * 注意:尽量不要在main方法上抛出异常
 *
 * 编译期异常抛出，调用者必须处理
 * 运行期异常，调用者可以不处理
 */
public class ExceptionDemo {
        public static void main(String[] args) {
                System.out.println("今天天气好");
                try {
                        method();
                } catch (ParseException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                System.out.println("有雾霾");// 能够输出

                method2();
        }

        // 运行期异常的抛出
        public static void method2() throws ArithmeticException {
                int a = 10;
                int b = 0;
                System.out.println(a / b);
        }
```

```
    public static void method() throws ParseException {
        String s = "2018-07-19";
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss");
        Date d = sdf.parse(s);
        System.out.println(d);
    }

}
```

# throw

2018年7月19日 14:37

throws
用在方法声明后，跟的是异常类名
可以跟多个异常类名，用逗号隔开
表示抛出异常，由该方法的调用者来处理
throws表示出现异常的一种可能性，并不一定会发生这些异常

throw：
用在方法体内，跟的是异常对象名
只能抛出一个异常对象名
表示抛出异常，由方法体内的语句处理
throw则是一定抛出某种异常

后续程序需要继续运行就用try
后续程序不需要继续运行就用throws

```
/*
 * throw:如果出现了异常情况，我们可以把该异常抛出，这是抛出的应该是异常的对象
 */
public class ExceptionDemo {
	public static void main(String[] args) {
		method();
		try {
			method2();
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public static void method() {
		int a = 10;
		int b = 0;
		if (b == 0) {
			throw new ArithmeticException();
		} else {
			System.out.println(a / b);
		}
	}

	public static void method2() throws Exception {
		int a = 10;
		int b = 0;
		if (b == 0) {
			throw new Exception();
		} else {
			System.out.println(a / b);
		}
	}
}
```

# finally

2018年7月19日 15:00

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/*
 * finally:被fianlly控制的语句体一定会执行
 * 注意：如果在执行到finally之前jvm退出了，既不能执行了
 * 格式：
 *                try...catch...finally
 * 作用：
 *                用于释放资源，在IO流和数据库操作中经常遇到
 */
public class FinallyDemo {
        public static void main(String[] args) {
                String s = "2018-07-19";
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
                Date d = null;
                try {
                        d = sdf.parse(s);
                } catch (ParseException e) {
                        e.printStackTrace();
                }finally {
                        System.out.println("可执行的代码");
                }

                System.out.println(d);
        }
}
```

# 面试题

2018年7月19日 15:14

1.final finally与finalize的区别

final：最终的意思，可以修饰类、成员变量、成员方法

修饰类:类不能被继承

修饰变量：变量是常量

修饰方法：方法不能被重写

finally：是异常处理的一部分，用于释放资源，代码肯定被执行

finalize：垃圾回收器调用，是Object类的方法

try...catch...finally的格式变形

1.try...catch...finally

2.try...catch

3.try...catch...catch

4.try...catch...catch...finally

5.try...finally

这种做法的目的是释放资源

2.如果catch里面有return语句，那么finally里面的代码还会执行吗

如果会，请问是在return前还是return后

```
public class FinallyDemo2 {
    public static void main(String[] args) {
        System.out.println(getInt());//30
    }

    public static int getInt() {
        int a = 0;
        try {
            System.out.println(a / 0);
            a = 20;
        } catch (ArithmeticException e) {
            a = 30;
            return a;
            /*
             * return a执行到这一步时，这里不是return a，而是return 30，
             * 这个返回路径就形成了，再继续执行finally的内容a=40，接着return 30
             */
        } finally {
            a = 40;
            //return a;//如果这样就是40
        }
        return a;
    }
}
```

# 自定义异常

2018年7月19日 15:57

```
/*
 * 为满足特定的需求，需要自定义异常
 * 自定义异常类必须继承自Exception或者RuntimeException
 */
public class MyException extends Exception {
	public MyException() {}

	public MyException(String message) {
		super(message);
	}
}
```

```
public class Teacher {
	public void check(int score) throws MyException {
		if(score>100 || score<0) {
			throw new MyException("分数越界");
		}else {
			System.out.println("分数没有问题");
		}
	}
}
```

```
import java.util.Scanner;

public class StudentDemo {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("请输入成绩");
		int score =sc.nextInt();

		Teacher t =new Teacher();
		try {
			t.check(score);
		} catch (MyException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

# 异常注意事项

2018年7月19日 16:26

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/*
 * 异常注意事项;
 *              A.子类重写父类方法时，子类的方法必须抛出相同的异常或父类异常的子类
 *              B.如果父类抛出多个异常，子类重写父类时，
 *              只能抛出相同的异常或其子类，子类不能抛出父类没有的异常
 *              C.如果被重写的方法没有异常抛出，子类绝不肯抛出异常，
 *              如果子类有异常，只能用try，不能用throws
 */
public class ExceptionDemo {

}

class Fu {
    public void show() throws ArithmeticException {

    }

    public void methow() {
    }
}

class Zi extends Fu {
    // public void show() throws Exception{ //不能是父类
    public void show() throws ArithmeticException {

    }

    public void methow() {
        String s = "2018-07-11";
        SimpleDateFormat sdf = new SimpleDateFormat();
        Date d = null;
        try {
```

```
            d = sdf.parse(s);
        } catch (ParseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } // 不能抛出异常（父类没抛）
        System.out.println(d);
    }
}
```

# File类

2018年7月19日 16:40

```
import java.io.File;

/*
 * File:是文件和路径名的抽象表达形式
 * 构造方法：(效果相同)
 *                File(String pathname):路径构造（常用）
 *                File(String pathname,String child)根据一个目录和一个子文件得到File对象
 *                File(File parent,String child)根据一个父file对象和子文件得到File对象
 */
public class FileDemo {
	public static void main(String[] args) {
		// File(String pathname)
		File file = new File("C:\\Users\\TJtulong\\Desktop\\a.txt");
		// 并不是创建，只是抽象的形式

		// File(String pathname,String child)
		File file2 = new File("C:\\Users\\TJtulong\\Desktop", "a.txt");

		// File(File parent,String child)
		File file3 = new File("C:\\Users\\TJtulong\\Desktop");
		File file4 = new File(file3, "a.txt");
	}
}
```

# 创建功能

2018年7月19日 22:08

```
import java.io.File;
import java.io.IOException;

/*
 * 创建功能:
 *                public boolean createNewFile():创建文件
 *                public boolean mkdir():创建文件夹,如果存在就不创建了，返回false
 *                public boolean mkdirs():创建文件夹,如果父文件夹不存在，会自动创建
 */
public class FileDemo {
        public static void main(String[] args) throws IOException {
                // 在桌面创建文件夹demo
                File file = new File("c:\\Users\\TJtulong\\Desktop\\demo");
                System.out.println("mkdir:" + file.mkdir());

                // 在demo文件夹下创建a.txt
                File file2 = new File("c:\\Users\\TJtulong\\Desktop\\demo\\a.txt");
                System.out.println("createnewfile:" + file2.createNewFile());

                // 在桌面text目录下创建b.txt
                // Exception in thread "main" java.io.IOException
                // 先建目录，在创造文件
                // File file3=new File("c:\\Users\\TJtulong\\Desktop\\test\\b.txt");
                // System.out.println("createnewfile:"+file3.createNewFile());

                // 在桌面test目录下创建aaa目录
                File file4 = new File("c:\\Users\\TJtulong\\Desktop\\test\\aaa");
                System.out.println("mkdir:" + file4.mkdir());// mkdir:false

                /*
                 * File file5=new File("c:\\Users\\TJtulong\\Desktop\\test"); File file6=new
                 * File("c:\\Users\\TJtulong\\Desktop\\test\\aaa");
                 * System.out.println("mkdir:"+file5.mkdir());
                 * System.out.println("mkdir:"+file6.mkdir());
                 */
```

```
        File file7 = new File("c:\\Users\\TJtulong\\Desktop\\test\\aaa");
        System.out.println("mkdir:" + file7.mkdirs());

        File file8 = new File("c:\\Users\\TJtulong\\Desktop\\test\\a.txt");
        System.out.println("mkdir:" + file8.mkdirs());
        // 创建了a.txt文件夹
    }
}
```

# 删除功能

2018年7月19日 22:39

```
import java.io.File;
import java.io.IOException;

/*
 * 注意：如果创建文件或目录忘记写盘符路径，默认在项目路径上
 * 删除功能：public boolean delete();
 * 注意：java中的删除不走回收站
 *                 要删除一个文件夹时，该文件夹内不能有文件或文件夹
 */
public class FileDemo {
	public static void main(String[] args) throws IOException {
		// 删除功能
		File file = new File("a.txt");
		file.createNewFile();
		System.out.println("delete:" + file.delete());

		File file2 = new File("c:\\Users\\TJtulong\\Desktop\\test\\aaa");
		file2.delete();
	}
}
```

# 重命名功能

2018年7月19日 23:00

```
import java.io.File;

/*
 * 重命名功能：public boolean renameTo(File dest)
 * 路径以盘符开始:绝对路径
 * 路径不以盘符开始：相对路径
 * 如果路径名相同就是改名，如果路径名不同就是剪切+改名
 */
public class FileDemo {
	public static void main(String[] args) {
		File file =new File("裂缝图1.jpg");

		//修改文件名为裂缝图
		File newfile = new File("裂缝图.jpg");
		file.renameTo(newfile);

		File newfile2 = new File("c:\\Users\\TJtulong\\Desktop\\裂缝图.jpg");
		System.out.println("rename:"+newfile.renameTo(newfile2));
		//文件出现在桌面上

	}
}
```

# 判断功能

2018年7月19日 23:37

```
import java.io.File;

/*
 * 判断功能:
 *                public boolean isDictionary():判断是否是目录
 *                public boolean isFile():判断是否是文件
 *                public boolean exists():判断是否是存在
 *                public boolean canRead():判断是否可读
 *                public boolean canWrite():判断是否可写
 *                public boolean isHidden();判断是否隐藏
 */
public class FileDemo {
	public static void main(String[] args) {
		File file = new File("a.txt");
		System.out.println(file.isDirectory());// false
		System.out.println(file.isFile());// true
		System.out.println(file.exists());// true
		System.out.println(file.canRead());// true
		System.out.println(file.canWrite());// true只读
		System.out.println(file.isHidden());// false隐藏
	}
}
```

# 普通获取功能

2018年7月19日 23:45

```
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;

/*
 * 获取功能：
 *                public String getAbsolutePath():获取绝对路径
 *                public String getPath()：获取相对路径
 *                public String getName()：获取名称
 *                public long length()：获取长度（字节）
 *                public long lastModified()：获取上一次的修改实现，返回毫秒值
 */
public class FileDemo {
        public static void main(String[] args) {
                File file = new File("demo\\test.txt");

                System.out.println(file.getAbsolutePath());
                System.out.println(file.getPath());
                System.out.println(file.getName());
                System.out.println(file.length());
                System.out.println(file.lastModified());

                Date d= new Date(file.lastModified());
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy--MM-dd
                HH:mm:ss");
                String s = sdf.format(d);
                System.out.println(s);
        }
}
```

# 高级获取功能

2018年7月20日 22:48

```
/*
 * 获取功能：
 *		public String[] list():获取指定目录下的所有文件或者文件夹的名称数组
 *		public File[] listFiles():获取指定目录下的所有文件或者文件夹的File数组
 */
public class FileDemo {
	public static void main(String[] args) {
		File file = new File("c:\\Users\\TJtulong\\Desktop");

		String[] strArray=file.list();
		for(String s:strArray) {
			System.out.println(s);
		}
		System.out.println("------------");

		File[] fileArray = file.listFiles();
		for(File f:fileArray) {
			System.out.println(f.getName());
		}
	}
}
```

# 练习

2018年7月20日 22:56

```
/*
 * 判断E盘下是否有后缀名为.jpg的文件，如果有输出文件名
 */
public class FileDemo {
	public static void main(String[] args) {
		File file = new File("c:\\Users\\TJtulong\\Desktop");

		File[] fileArray = file.listFiles();
		for (File f : fileArray) {
			if (f.isFile()) {
				if (f.getName().endsWith(".pdf")) {
					System.out.println(f.getName());
				}
			}
		}
	}
}
```

# 文件名过滤器

2018年7月20日 23:40

```
import java.io.File;
import java.io.FilenameFilter;

/*
 * 判断E盘下是否有后缀名为.jpg的文件，如果有输出文件名
 * 文件名过滤器
 * public String[] list(FileNameFilter filter)
 * public File[] listFiles(FileNameFilter filter)
 */
public class FileDemo2 {
	public static void main(String[] args) {
		File file = new File("c:\\Users\\TJtulong\\Desktop");

		String[] strArray = file.list(new FilenameFilter() {

			@Override
			public boolean accept(File dir, String name) {
				// return false;
				File file = new File(dir, name);
				boolean flag = file.isFile();
				boolean flag2 = name.endsWith(".pdf");
				return flag && flag2;
			}
		});

		for (String s : strArray) {
			System.out.println(s);
		}
	}
}
```