Java进阶1 第5天

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【应用】异常概述与分类**
2. 【理解】能够阐述异常的由来
3. 【应用】能够独立写出异常的继承体系
4. **【应用】异常的处理方式**
5. 【应用】能够使用try…catch处理单个异常和多个异常
6. 【应用】能够使用 throws 处理单个异常和多个异常
7. **【应用】异常的注意事项与自定义异常**
8. 【理解】能够阐述异常在继承中的注意事项
9. 【理解】能够阐述当有多个catch,catch中参数为父类的异常需要放在最后的原因
10. 【理解】能够阐述写在finally中代码的作用
11. 【理解】能够独立定义一个自定义异常,并使用该异常

# 异常概述与分类

## 异常概念

A:异常:就是程序运行中出现的不正常的情况.

B:异常的由来:程序中的问题也是现实生活中的一个具体的事物,也可以通过java的类的形式进行描述,并封装成对象.

其实异常就是java对不正常情况描述后的对象体现.

## 异常的分类

A:异常的继承体系

Throwable

Error (错误:一出现就是致命的)

服务器宕机,数据库崩溃等

Exception

RuntimeException(运行时异常,一般都是程序员犯的错误,需要修改源码的.)

编译时异常:在编译时必须进行处理,不处理无法通过编译.

### 案例代码一:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: ExceptionDemo  \* **@Description**: 异常  \* **@date** 2017年11月16日 上午9:17:26  \* Company www.igeekhome.com  \*  \* 概念:代码运行当中可能出现的不正常情况  \* java当中将这些已经预见到的不正常情况,封装成了类.即异常类.  \*  \* 产生异常:  \* 当满足了异常产生条件时,会执行以下两个动作  \* 创建异常对象  \* 抛出异常对象  \* 执行完该两个动作后,叫做产生了异常,只是这个过程是java自动执行的,我们看不到.  \*  \* 异常分类:  \* Throwable  \* Error(程序无法处理)  \* Exception(可以通过程序处理的)  \* 编译期异常(编译过程报错,必须处理)  \* 运行期异常RuntimeException(编译期可以不处理,但是运行时可能会报错)  \*  \* 处理:  \* 捕获处理  \* 声明抛出处理  \*  \*/  **public** **class** ExceptionDemo {  **public** **static** **void** main(String[] args) **throws** IOException {    //编译时异常  //"d:/test.txt"  //指定文件  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入文件路径：");  String path = sc.next();    FileWriter fw = **new** FileWriter(path);  //写出内容  //fw.write("i love java");  //关闭资源  //fw.close();  System.***out***.println("后边的代码");    //运行时异常  System.***out***.println(10 / 0);  }  } |

# 异常的处理方式

## try…catch处理单个异常

### try…catch格式:

|  |
| --- |
| try{  需要检测的异常；  } catch(异常类名异常变量) {  异常处理代码  可以调用异常的方法  通常我们只使用一个方法：printStackTrace打印异常信息  } |

### 案例代码二:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: TryCatchDemo  \* **@Description**: 异常捕获处理  \* **@date** 2017年11月16日 上午9:28:05  \* Company www.igeekhome.com  \*  \* 捕获处理，将异常获取，使用try/catch做分支处理  \*  \* try{  \* 需要检测的异常；  \* } catch(异常对象) {  \* 异常处理代码  \* 可以调用异常的方法  \* 通常我们只使用一个方法：printStackTrace打印异常信息  \* }  \*/  **public** **class** TryCatchDemo {  **public** **static** **void** main(String[] args) {  //编译时异常  //"d:/test.txt"  //指定文件  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入文件路径：");  String path = sc.next();    **try** {  FileWriter fw = **new** FileWriter(path);  System.***out***.println("输出成功了");  } **catch**(IOException e) {  System.***out***.println("您输入的路径有误!");  //用于展示异常出现的位置和基本信息  e.printStackTrace();  }    System.***out***.println("后边的代码");  }  } |

## 使用throws关键字处理异常

throws的方式处理异常

定义功能方法时，需要把出现的问题暴露出来让调用者去处理。

那么就通过throws在方法上标识

### throws格式

修饰符 返回值类型 方法名(参数类型参数名1,参数类型参数名2...) **throws 异常类名** {

}

### 案例代码三:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: ThrowsDemo  \* **@Description**: 异常声明抛出处理  \* **@date** 2017年11月16日 上午9:35:25  \* Company www.igeekhome.com  \*  \* 在某个方法的声明上,让它(这个方法)抛出这个产生的异常,自己不处理,谁调用这个方法,谁处理  \* 方法声明上加 throws 异常类名  \*  \* 如:  \* 方法a中产生了异常,声明抛出  \* 方法b调用方法a,则该异常由a转向了b  \*  \* 一般情况下,main方法是不会声明抛出异常的,因为该方法抛出,直接抛出给了JVM,JVM的默认处理机制是终止程序运行,则这样处理,相当于没有处理!  \*  \*/  **public** **class** ThrowsDemo {  **public** **static** **void** main(String[] args) {  **try** {  *method*();  } **catch** (IOException e) {  System.***out***.println("文件有误");  //调用方法,打印异常信息  e.printStackTrace();  }    System.***out***.println("后边的代码");  }  /\*\*  \* **@Title**: method  \* **@Description**: 抛出异常的方法  \* **@throws** IOException  \*/  **public** **static** **void** method() **throws** IOException {  *method2*();  }    /\*  \* 该方法的代码中,可能会产生IOException异常,该方法处理的方式是声明如果出现该异常,就抛出该异常,自身并不处理,哪个方法调用该方法,哪个方法处理  \*/  /\*\*  \* **@Title**: method2  \* **@Description**: 抛出异常的方法2  \* **@throws** IOException  \*/  **public** **static** **void** method2() **throws** IOException {    //编译时异常  //"d:/test.txt"  //指定文件  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入文件路径：");  String path = sc.next();    FileWriter fw = **new** FileWriter(path);  System.***out***.println("输出成功了");  }  } |

## 单个运行时异常处理

### 案例代码四:

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: RuntimeExceptionDemo  \* **@Description**: 运行时异常处理  \* **@date** 2017年11月16日 上午9:42:39  \* Company www.igeekhome.com  \*  \* 该异常在编译期不检查语法错误的.可以处理,也可以不处理.  \*  \* 一般情况下,遇到java自身提供的运行时异常,都可以通过逻辑解决,避免异常产生.  \* 常见运行时异常:空指针异常,索引越界异常,类型转换异常  \*  \* 声明抛出处理  \* 捕获处理  \*/  **public** **class** RuntimeExceptionDemo {  **public** **static** **void** main(String[] args) {    **try** {  System.***out***.println(10/0);  } **catch** (ArithmeticException ae) {  //可以打印异常信息,也可以不打印  //ae.printStackTrace();    //如果出现异常的话  System.***out***.println("除数不能为0!");  }  System.***out***.println("后边的代码");    **try** {  *method*();  } **catch** (ArithmeticException ae) {  System.***out***.println("除数不能为0!");  }  }    /\*\*  \* **@Title**: method  \* **@Description**: 运算方法  \* **@throws** ArithmeticException  \*/  **public** **static** **void** method() **throws** ArithmeticException{  System.***out***.println(10/1);  }  } |

## 多个异常处理

### try…catch处理多个异常格式

|  |
| --- |
| try{  需要检测的异常；  } catch(异常类名1 异常变量1) {  异常处理代码  可以调用异常的方法  通常我们只使用一个方法：printStackTrace打印异常信息  } catch(异常类名2 异常变量2) {  异常处理代码  可以调用异常的方法  通常我们只使用一个方法：printStackTrace打印异常信息  }......//可以有无数个catch |

### throws 处理多个异常格式

修饰符 返回值类型 方法名(参数类型参数名1,参数类型参数名2...)

**throws 异常类名1,异常类名2,......**{

}

### 案例代码五:

|  |
| --- |
| **package** com.igeek\_04;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: MultiExceptionDemo  \* **@Description**: 多异常处理  \* **@date** 2017年11月16日 上午9:49:52  \* Company www.igeekhome.com  \*  \* 多异常处理一:  \* 声明抛出处理,直接抛出多个异常即可  \*  \* 多异常处理二:  \* 一个try多个catch的方式  \*  \* java是中断异常的处理方式  \*/  **public** **class** MultiExceptionDemo {  **public** **static** **void** main(String[] args) {    //一个try多个catch的方式  **try**{  *method*();  }**catch**(ArithmeticException ae) {  System.***out***.println("您的除数不能为0");  }**catch**(IOException e) {  System.***out***.println("您要访问的文件不存在");  }    System.***out***.println("后边的代码");  }  //一个方法声明抛出多个异常  /\*\*  \* **@Title**: method  \* **@Description**: 抛出异常的方法  \* **@throws** ArithmeticException  \* **@throws** IOException  \*/  **public** **static** **void** method() **throws** ArithmeticException,IOException{    System.***out***.println(10/1);  System.***out***.println("method逻辑1完成");    *method2*();  System.***out***.println("method逻辑2完成");  }    /\*\*  \* **@Title**: method2  \* **@Description**: 抛出异常的方法2  \* **@throws** IOException  \*/  **public** **static** **void** method2() **throws** IOException {  //指定文件  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入文件路径：");  String path = sc.next();    FileWriter fw = **new** FileWriter(path);  }  } |

### 多个异常用一个catch处理格式

try{

需要检测的异常；

} catch(**异常类名1|异常类名2……**异常变量) {

异常处理代码

可以调用异常的方法

通常我们只使用一个方法：printStackTrace打印异常信息

}

### 案例代码六:

|  |
| --- |
| **package** com.igeek\_04;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: MultiExceptionDemo  \* **@Description**: 多个异常用一个catch处理格式  \* **@date** 2017年11月16日 上午9:49:52  \* Company www.igeekhome.com  \*  \*  \* 1、多个异常分别每个try对应一个catch处理，不建议  \* 2、多个异常只有一个try一个catch  \* 3、一个try多个catch的方式  \*/  **public** **class** MultiExceptionDemo2 {  **public** **static** **void** main(String[] args) {    /\*多个异常分别处理  Scanner sc = new Scanner(System.in);  String path = sc.next();    try {  FileWriter fw = new FileWriter(path);  } catch (IOException e) {  System.out.println("您访问的文件不存在!");  }    try {  System.out.println(10/0);  } catch (ArithmeticException e) {  System.out.println("除数不能为0");  }  \*/    /\*多个异常一次try一次catch  Scanner sc = new Scanner(System.in);  String path = sc.next();    try {  FileWriter fw = new FileWriter(path);  System.out.println(10/0);  }catch(ArithmeticException|IOException ae) {  System.out.println("您访问的文件不存在!或者除数不能为0");  }    System.out.println("后边的代码");  \*/    //一个try多个catch的方式  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入文件路径：");  String path = sc.next();  **try**{  FileWriter fw = **new** FileWriter(path);  System.***out***.println(10/0);  }**catch**(ArithmeticException ae) {  System.***out***.println("您的除数不能为0");  }**catch**(IOException e) {  System.***out***.println("您要访问的文件不存在");  }    System.***out***.println("后边的代码");  }  } |

# 异常的注意事项与自定义异常

## 异常在继承中的注意事项

 A: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。(父亲坏了,儿子不能比父亲更坏)

b:如果父类抛出了多个异常,子类重写父类时,只能抛出相同的异常或者是他的子集,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,那么子类的方法绝对不可以抛出异常,如果子类方法内有异常发生,那么子类只能try,不能throws

### 案例代码七:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.IOException;  **import** java.text.ParseException;  **import** java.util.zip.DataFormatException;  /\*\*  \* **@ClassName**: MultiExceptionNotesDemo  \* **@Description**: 异常在继承中的注意事项  \* **@date** 2017年11月16日 上午10:21:59  \* Company www.igeekhome.com  \*  \* 如果父类抛出了多个异常,子类覆盖父类方法时,只能抛出相同的异常或者是他的子集,只能少不能多  \* 父类方法没有抛出异常，子类覆盖父类该方法时也不可抛出异常。此时子类产生该异常，只能捕获处理，不能声明抛出  \*  \* 现实开发过程中,一般子类重写的方法抛出的异常与父类方法抛出的异常是一样的,不存在不一样的情况.  \*/  **public** **class** MultiExceptionNotesDemo {  // 为了授课方便,则在一个文件中定义多个类,其他类不能使用public修饰    // 如果父类抛出了多个异常,子类覆盖父类方法时,只能抛出相同的异常或者是他的子集  /\*\*  \* **@ClassName**: Fu  \* **@Description**: 父类  \* **@date** 2017年11月16日 上午10:40:05  \* Company www.igeekhome.com  \*  \*/  **class** Fu {  /\*\*  \* **@Title**: method  \* **@Description**: 父类中抛出异常的方法  \* **@throws** IOException  \* **@throws** ParseException  \* **@throws** DataFormatException  \*/  **public** **void** method() **throws** IOException,ParseException,DataFormatException{  //产生异常的方法体  }  }    /\*\*  \* **@ClassName**: Zi  \* **@Description**: 子类  \* **@date** 2017年11月16日 上午10:40:15  \* Company www.igeekhome.com  \*  \*/  **class** Zi **extends** Fu {    /\*\*  \* **@Title**: method  \* **@Description**: 子类中重写父类的方法  \* **@throws** IOException  \* **@throws** ParseException  \* **@see** javaadvanced1\_d03\_c03.MultiExceptionNotesDemo.Fu#method()  \*/  @Override  **public** **void** method() **throws** IOException, ParseException {  //子类的方法体  }  }  } |

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: MultiExceptionNotesDemo  \* **@Description**: 异常在继承中的注意事项2  \* **@date** 2017年11月16日 上午10:21:59  \* Company www.igeekhome.com  \*  \* 如果父类抛出了多个异常,子类覆盖父类方法时,只能抛出相同的异常或者是他的子集,只能少不能多  \* 父类方法没有抛出异常，子类覆盖父类该方法时也不可抛出异常。此时子类产生该异常，只能捕获处理，不能声明抛出  \*  \* 现实开发过程中,一般子类重写的方法抛出的异常与父类方法抛出的异常是一样的,不存在不一样的情况.  \*/  **public** **class** MultiExceptionNotesDemo2 {  // 为了授课方便,则在一个文件中定义多个类,其他类不能使用public修饰    // 父类方法没有抛出异常，子类覆盖父类该方法时也不可抛出异常。此时子类产生该异常，只能捕获处理，不能声明抛出  /\*\*  \* **@ClassName**: Fu  \* **@Description**: 父类  \* **@date** 2017年11月16日 上午10:41:38  \* Company www.igeekhome.com  \*  \*/  **class** Fu {  /\*\*  \* **@Title**: method  \* **@Description**: 父类中的方法  \*/  **public** **void** method() {  // 不产生异常的方法体  }  }  /\*\*  \* **@ClassName**: Zi  \* **@Description**: 子类  \* **@date** 2017年11月16日 上午10:41:50  \* Company www.igeekhome.com  \*  \*/  **class** Zi **extends** Fu {  /\*\*  \* **@Title**: method  \* **@Description**: 子类重写父类的方法  \* **@see** javaadvanced1\_d03\_c03.MultiExceptionNotesDemo2.Fu#method()  \*/  @Override  **public** **void** method() {  // 子类的方法体  // 编译时异常  //"d:/test.txt"  //指定文件  Scanner sc = **new** Scanner(System.***in***);  String path = sc.next();  **try** {  FileWriter fw = **new** FileWriter(path);  } **catch** (IOException e) {  System.***out***.println("文件不存在");  }  }  }  } |

## 处理多个构成子父类关系的异常

### 案例代码八:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: MultiExceptionNotesDemo  \* **@Description**: catch多个异常时的注意事项  \* **@date** 2017年11月16日 上午10:21:59  \* Company www.igeekhome.com  \*  \* 在捕获异常时,可以使用父类异常的引用指向子类异常的对象.是多态的应用.  \* 当多异常处理时，捕获处理，前边的类不能是后边类的父类  \*/  **public** **class** MultiExceptionNotesDemo {  **public** **static** **void** main(String[] args) {  //一个try多个catch的方式  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入文件路径：");  String path = sc.next();    **try**{  System.***out***.println(10/0);  FileWriter fw = **new** FileWriter(path);  }**catch**(IOException e) {  System.***out***.println("您要访问的文件不存在");  }**catch**(Exception ae) {  System.***out***.println("您的除数不能为0");  }    System.***out***.println("后边的代码");    /\*多个异常一次try一次catch,异常之间不能存在子父类关系    Scanner sc = new Scanner(System.in);  String path = sc.next();    try {  FileWriter fw = new FileWriter(path);  System.out.println(10/0);  }catch(IOException|Exception ae) {  System.out.println("您访问的文件不存在!或者除数不能为0");  }    System.out.println("后边的代码");  \*/  }  } |

## finally关键字

|  |
| --- |
| try{  需要检测的异常；  } catch(异常类名异常变量) {  异常处理代码  可以调用异常的方法  通常我们只使用一个方法：printStackTrace打印异常信息  } finally{  此处存放一定要执行的代码  一般把释放资源的代码放到finally中  } |

### 案例代码九:

|  |
| --- |
| **package** com.igeek\_03;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: FinallyDemo  \* **@Description**: finally代码块  \* **@date** 2017年11月16日 上午10:55:49  \* Company www.igeekhome.com  \*  \* 在try/catch后可以追加finally代码块，其中的代码一定会被执行，通常用于资源回收。  \*/  **public** **class** FinallyDemo {  **public** **static** **void** main(String[] args) {    //编译时异常  //"d:/test.txt"  //指定文件  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入文件路径：");  String path = sc.next();    **try** {  FileWriter fw = **new** FileWriter(path);  fw.write("abc");  } **catch** (IOException e) {  System.***out***.println("产生了IO问题");  } **finally** {  System.***out***.println("我要关闭资源");  }    System.***out***.println("后边的代码");  }  } |

## 自定义异常

在上述代码中，发现这些异常都是JDK内部定义好的，并且这些异常不好找。书写时也很不方便，那么能不能自己定义异常呢？

之前的几个异常都是java通过类进行的描述，并将问题封装成对象，异常就是将问题封装成了对象。

这些异常不好认，书写也很不方便，能不能定义一个符合我的程序要求的异常名称。

既然JDK中是使用类在描述异常信息，那么我们也可以模拟Java的这种机制，我们自己定义异常的信息，异常的名字，让异常更符合自己程序的阅读。

准确对自己所需要的异常进行类的描述

### 自定义异常格式

Class 异常名 extends Exception{ **//或继承RuntimeException**

public 异常名(){

}

public 异常名(String s){

super(s);

}

}

### 案例代码十:

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: U18Exception  \* **@Description**: 小于18岁异常  \* **@date** 2017年11月16日 上午11:01:20  \* Company www.igeekhome.com  \*  \*/  **public** **class** U18Exception **extends** Exception {    /\*\*  \* **@Title**: U18Exception  \* **@param** message  \*/  **public** U18Exception(String message) {  **super**(message);  }  } |

|  |
| --- |
| **package** com.igeek\_04;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: CustomExceptionDemo  \* **@Description**: 自定义异常的测试类  \* **@date** 2017年11月16日 上午11:02:10  \* Company www.igeekhome.com  \*  \* 自定义异常:  \* 编译时异常继承Exception，运行时异常继承RuntimeException。  \* 关键字 throw 抛出异常  \* 回忆产生异常:  \* 创建异常对象  \* 抛出异常  \*/  **public** **class** CustomExceptionDemo {  **public** **static** **void** main(String[] args) {  //年龄小于18岁,产生异常  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入年龄：");  **int** age = sc.nextInt();    **if**(age < 18) {  //产生异常  //创建异常对象  U18Exception u18 = **new** U18Exception("年龄小于18岁");  //抛出异常  **try** {  **throw** u18;  } **catch** (U18Exception e) {  System.***out***.println("年龄不满足婚恋网站年龄");  e.printStackTrace();  }  }    //可以直接手动创建异常对象  **try** {  **throw** **new** RuntimeException("我产生了一个运行时异常");  } **catch** (Exception e) {  System.***out***.println("直接创建了一个RuntimeException对象");  }    System.***out***.println("后边的代码");  }  } |

重点和总结

1、异常的概述与分类 ：编译时异常、运行时异常

2、各种异常形式的不同处理方式

3、异常处理中的相关关键字：try、catch、throw、throws、finally

4、异常处理中的相关注意事项

5、自定义异常