第5章 异　常

5.1 异常概述

5.1.1 什么是异常

生活中经常会遇到一些不正常的现象，比如人会生病、机器会坏、计算机会死机等。而你写的代码也并不是完美的，比如要读取一个文件，如果这个文件不存在该怎么办，如果这个文件不可读又该怎么办，等等。程序在运行过程中可能出现的这些不正常现象就称作异常。异常(Exception)，意思是例外，怎么让写的程序做出合理的处理，安全地退出，而不至于程序崩溃。

Java 是采用面向对象的方式处理异常的。当程序出现问题时，就会创建异常类对象并抛出异常相关的信息(如异常出现的位置、原因等)，从而能够更迅速地定位到问题原因。

5.1.2 异常与错误

异常与错误是很容易混淆的两个概念，异常指的是程序运行过程中出现的不正常现象，比如文件读不到、链接打不开，影响了正常的程序执行流程，但不至于程序崩溃，这些情况对于程序员而言是可以处理的。而错误则是程序脱离了程序员的控制，一般指程序运行时遇到的硬件或操作系统的错误，如内存溢出、不能读取硬盘分区、硬件驱动错误等。错误是致命的，将导致程序无法运行，同时也是程序本身不能处理的。

比如代码清单 5.1 就分别演示了异常和错误的区别。

代码清单 5.1　Demo1Exception

```
package com.yyds.unit5.demo;
public class Demo1Exception {
    public static void main(String[] args) {
```

```
        //除数为 0,抛出异常。这种情况程序员可以控制,把计算时除数为 0 的情况排除即可
          int num = 3 / 0;
          //内存溢出,这是错误。这种情况程序员无法处理,因为内存溢出可能与硬件设备相关,
          //比如计算机内存只有 256MB
          int[] arr = new int[1024 * 1024 * 1024];
    }
}
```

分别运行异常和错误的两行代码,程序(代码清单 5.1)运行结果如图 5.1 所示。

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at com.yyds.unit5.demo.Demo1Exception.main(Demo1Exception.java:5)    异常
Exception in thread "main" java.lang.OutOfMemoryError Create breakpoint : Java heap space
    at com.yyds.unit5.demo.Demo1Exception.main(Demo1Exception.java:7)    错误
```

图 5.1　程序运行结果

5.1.3　Throwable 与异常体系

Java 中,异常(Exception)与错误(Error)都继承自 Throwable 类。Java 中定义了大量的异常类,这些类对应了各种各样可能出现的异常事件,这些异常类都直接或间接地继承了 Exception。Exception 分为运行时异常(RuntimeException)和编译时异常,Error 和 RuntimeException 由于在编译时不会进行检查,因此又称为不检查异常(UncheckedException),而编译时异常会在编译时进行检测,又称为可检查异常(CheckedException)。

Java 中的异常体系结构如图 5.2 所示。

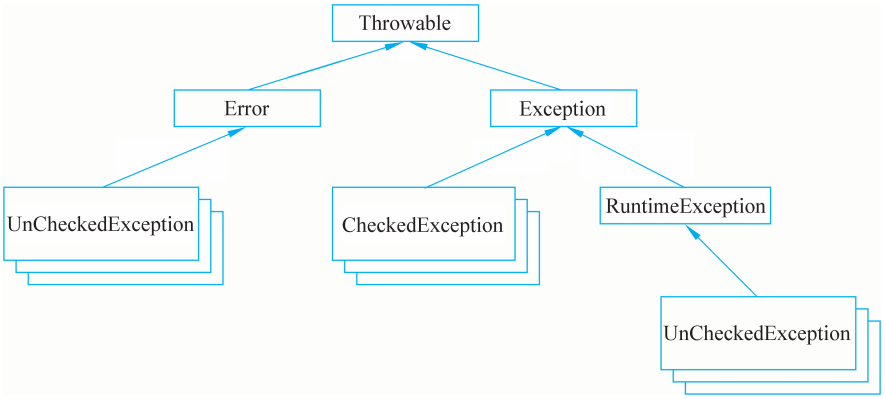

图 5.2　Java 中的异常体系结构

Throwable 中定义了所有异常都会用到的 3 个重要方法,如表 5.1 所示。

表 5.1　Throwable 主要方法

方 法 签 名	方 法 描 述
String getMessage()	返回此 Throwable 的详细消息字符串
String toString()	返回此 Throwable 的简短描述
void printStackTrace()	打印异常的堆栈跟踪信息

5.1.4 Exception

Exception 是所有异常的父类，其本身是编译时异常，而它的一个子类 RuntimeException 则是运行时异常。

RuntimeException 和它的所有子类都是运行时异常，比如 NullPointerException、ArrayIndexOutOfBoundsException 等，这些异常在程序编译时不能被检查出，往往是由逻辑错误引起的，因此在编写程序时，应该从逻辑的角度尽可能避免这种异常情况，如代码清单 5.2 所示，这个程序就是因为没有对用户输入的值进行判断，从而导致出现了异常。

代码清单 5.2　Demo2RuntimeException

```
package com.yyds.unit5.demo;
import java.util.Scanner;
public class Demo2RuntimeException {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] arr = {"北京", "上海", "武汉", "广州", "深圳"};
        System.out.println("请输入要获取的城市索引");
        int index = scanner.nextInt();
        System.out.println("您获取的城市为: " + arr[index]);
    }
}
```

程序(代码清单 5.2)运行结果如图 5.3 所示。

```
请输入要获取的城市索引
6
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : 6
    at com.yyds.unit5.demo.Demo2RuntimeException.main(Demo2RuntimeException.java:11)
```

图 5.3　程序运行结果

Exception 和它的子类(不包括 RuntimeException 及其子类)统称为编译时异常，比如 IOException、SQLException。这些异常在编译时会被检查出，程序员必须对其进行处理，如代码清单 5.3 就是典型的编译时异常，如果不对这个异常进行处理，程序编译就不会通过。

代码清单 5.3　Demo3CheckedException

```
package com.yyds.unit5.demo;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
public class Demo3CheckedException {
    public static void main(String[] args) throws FileNotFoundException {
        //I/O 流,后面章节会涉及
        InputStream is = new FileInputStream("D:\\abc.txt");
    }
}
```

5.2 异常处理


扫一扫

5.2.1 抛出异常

编写程序时，需要考虑程序可能出现的各种问题，比如编写一个方法，对于方法中的参数就需要进行一定程度的校验。如果参数校验通不过，需要告诉调用者问题原因所在，这时候就需要抛出异常。

Java 中提供了一个 throw 关键字，该关键字用于抛出异常。Java 中的异常本质上也是类，抛出异常时，实际上是抛出一个异常的对象，并提供异常文本给调用者，最后结束该方法的运行。抛出异常的语法格式如下。

```
throw new 异常名称(参数列表);
```

比如对于数组的操作，可以定义一个方法用来获取数组指定索引的值，当索引不合法时，通过抛出异常的方式通知调用者，如代码清单 5.4 所示。

代码清单 5.4　Demo4Throw

```
package com.yyds.unit5.demo;
import java.util.Scanner;
public class Demo4Throw {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] arr = {"北京", "上海", "武汉", "广州", "深圳"};
        System.out.println("请输入要获取的城市索引");
        int index = scanner.nextInt();
        String value = getValue(arr, index);
        System.out.println("您获取的城市为: " + value);
    }
    private static String getValue(String[] arr, int index) {
        //对参数校验
        if(arr == null) {
            //抛出非法参数异常,并指定提示文本
            throw new IllegalArgumentException("数组不能为空!");
        }
        if(index < 0) {
            throw new IllegalArgumentException("索引不能为负数!");
        }
        if(index >= arr.length) {
            //抛出数组索引越界异常
            throw new ArrayIndexOutOfBoundsException("索引不可以超过数组长度!");
        }
        return arr[index];
    }
}
```

运行程序，输入一个非法的参数，比如索引为－1，程序(代码清单 5.4)运行结果如图 5.4 所示。

```
请输入要获取的城市索引
-1
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : 索引不能为负数!
    at com.yyds.unit5.demo.Demo4Throw.getValue(Demo4Throw.java:21)
    at com.yyds.unit5.demo.Demo4Throw.main(Demo4Throw.java:11)
```

图 5.4　程序运行结果

可以看到，当参数通不过校验时，就会执行抛出异常的代码，并且提示文本也是在代码中定义好的。

5.2.2　声明异常


扫一扫

程序仅仅抛出异常，而不对异常进行处理，是没有任何意义的，但处理异常之前，需要知道调用的方法可能会抛出哪些异常，从而有针对性地处理。因此，需要将异常声明到方法上，让调用者知道这个方法可能会抛出什么异常。

声明异常使用 throws 关键字(与 throw 非常像，需要注意)，语法格式如下。

```
修饰符 返回值类型 方法名(参数列表) throws 异常类名 1, 异常类名 2, … {  }
```

其中，如果方法中抛出的是运行时异常，编译期就不会强制要求开发者将异常声明在方法上，而如果抛出的是编译时异常，则必须将这些异常全部声明在方法上。比如上面的程序中，就可以将异常声明到方法上，如代码清单 5.5 所示。

代码清单 5.5　Demo5Throws

```
package com.yyds.unit5.demo;
import java.util.Scanner;
public class Demo5Throws {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] arr = {"北京", "上海", "武汉", "广州", "深圳"};
        System.out.println("请输入要获取的城市索引");
        int index = scanner.nextInt();
        String value = getValue(arr, index);
        System.out.println("您获取的城市为: " + value);
    }
    //在方法上声明异常
    private static String getValue(String[] arr, int index)
            throws IllegalArgumentException, ArrayIndexOutOfBoundsException {
        //参数校验
        if(arr == null) {
            //抛出非法参数异常,并指定提示文本
            throw new IllegalArgumentException("数组不能为空!");
        }
        if(index < 0) {
            throw new IllegalArgumentException("索引不能为负数!");
```

```
        }
        if(index >= arr.length) {
            //抛出数组索引越界异常
            throw new ArrayIndexOutOfBoundsException("索引不可以超过数组长度!");
        }
        return arr[index];
    }
}
```


扫一扫

5.2.3 捕获异常

如果程序出现了异常，自己又解决不了，就需要将异常声明出来，交给调用者处理。上面已经将异常进行了声明，此时调用者如果已经知道被调用方法可能出现哪些异常，就可以针对这些异常进行不同的处理。

处理异常使用 try…catch…finally 结构，try 用于包裹住可能出现异常的代码块，在 catch 中进行异常捕获，并处理异常，finally 则是在抛出异常或者异常处理后执行。当异常不进行处理时，发生异常的方法就会立即结束运行，而如果使用 try…catch 处理，程序就会继续运行下去。接下来再对上面的代码进行修改，对两个异常进行处理，如代码清单 5.6 所示。

代码清单 5.6　Demo6TryCatch

```
package com.yyds.unit5.demo;
import java.util.Scanner;
public class Demo6TryCatch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] arr = {"北京", "上海", "武汉", "广州", "深圳"};
        System.out.println("请输入要获取的城市索引");
        int index = scanner.nextInt();
        try {
            //可能出现异常的代码,用 try 包裹住
            String value = getValue(arr, index);
            System.out.println("您获取的城市为: " + value);
        }catch(IllegalArgumentException e) {
            //这个 catch 中处理 IllegalArgumentException
            //输出异常文本
            System.out.println("程序发生了非法参数异常: "+e.getMessage());
        }catch(ArrayIndexOutOfBoundsException e) {
            //这个 catch 中处理 ArrayIndexOutOfBoundsException
            //打印异常
            e.printStackTrace();
        }finally {
            //不管是否发生异常,最终都会执行 finally
            System.out.println("程序运行完毕");
        }
    }
```

```
    //在方法上声明异常
    private static String getValue(String[] arr, int index)
            throws IllegalArgumentException, ArrayIndexOutOfBoundsException {
        //对参数进行校验
        if(arr == null) {
            //抛出非法参数异常,并指定提示文本
            throw new IllegalArgumentException("数组不能为空!");
        }
        if(index < 0) {
            throw new IllegalArgumentException("索引不能为负数!");
        }
        if(index >= arr.length) {
            //抛出数组索引越界异常
            throw new ArrayIndexOutOfBoundsException("索引不可以超过数组长度!");
        }
        return arr[index];
    }
}
```

程序(代码清单 5.6)运行结果如图 5.5 所示。

```
请输入要获取的城市索引
-1
程序发生了非法参数异常：索引不能为负数!
程序运行完毕
请输入要获取的城市索引
5
java.lang.ArrayIndexOutOfBoundsException Create breakpoint : 索引不可以超过数组长度!
    at com.yyds.unit5.demo.Demo6TryCatch.getValue(Demo6TryCatch.java:41)
    at com.yyds.unit5.demo.Demo6TryCatch.main(Demo6TryCatch.java:13)
程序运行完毕
请输入要获取的城市索引
3
您获取的城市为：广州
程序运行完毕
```

图 5.5 程序运行结果

一个 try 必须跟随至少一个 catch 或者 finally 关键字,即 try…catch 结构、try…finally 结构和 try…catch…finally 结构都是合法的。异常的处理是链式的,如果存在 main->methodA->methodB->methodC 的调用链,此时 methodC 发生了异常,就会抛给 methodB 处理。如果 methodB 处理不了或者没有处理,就会再抛给 methodA 处理,如果 methodA 依然没法处理,就会抛给 main()方法处理,也就是交给虚拟机处理,而虚拟机处理异常的方式简单、粗暴:直接打印异常堆栈信息并停止程序的运行。因此,在开发中为了防止死机,程序能进行处理的异常要尽可能交由程序处理。

此外,如果一个异常并没有在方法上声明,则在代码中依然可以捕获这个异常。声明异常只是为了方便开发者知道要处理哪些异常。

5.3 异常进阶


扫一扫

5.3.1 自定义异常

尽管 Java 中已经定义了大量的异常，但实际开发中这些异常并不能完全涵盖所有的业务场景，不能通过异常文本判断业务逻辑问题所在。

例如，登录场景中，绝大多数网站为了防止暴力撞库，对于用户不存在和密码错误两种场景的提示文本都是“用户名或密码错误”，这是为了防止不法分子根据提示文本而推断出网站的用户。

但是，对于开发者而言，需要在日志中能够区分某个“用户名或密码错误”究竟是用户不存在，还是密码错误，此时就需要 UserNotFoundException 和 PasswordWrongException 两个异常。很明显，Java 中不可能事先定义好这些异常，因此就需要用户自定义一些与业务场景相关的异常。

自定义异常语法很简单，就是创建一个类并继承 Exception 或者 RuntimeException，提供相应的构造方法，如下所示。

```
修饰符 class 自定义异常名 extends Exception 或 RuntimeException {
    public 自定义异常名() {
        //默认调用父类无参构造方法
    }
    public 自定义异常名(String msg) {
        //调用父类具有异常信息的构造方法
        super(msg);
    }
}
```

大多数情况下，不需要在构造方法中定义其他逻辑，直接调用父类异常对应的构造方法即可，自定义异常就是这么简单。

下面编写一个登录案例，用户输入用户名和密码，对于用户名不存在和密码错误两种场景，均提示“用户名或密码错误”，但抛出异常时则抛出不同的异常。

首先定义 UserNotFoundException 和 PasswordWrongException 两个异常类，如下所示。

```
package com.yyds.unit5.demo;
public class UserNotFoundException extends Exception{
    public UserNotFoundException() {
    }
    public UserNotFoundException(String message) {
        super(message);
    }
}
```

```
package com.yyds.unit5.demo;
public class PasswordWrongException extends RuntimeException{
    public PasswordWrongException() {
    }
    public PasswordWrongException(String message) {
        super(message);
    }
}
```

尽管它们继承的异常不同,但内部的代码都是一模一样的。接下来编写登录逻辑,如代码清单 5.7 所示。

代码清单 5.7　Demo7Login

```
package com.yyds.unit5.demo;
import java.util.Scanner;
public class Demo7Login {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入用户名: ");
        String username = scanner.next();
        System.out.print("请输入密码: ");
        String password = scanner.next();
        try {
            login(username, password);
            //如果方法成功运行完毕,说明两个异常都没有触发,登录成功
            System.out.println("登录成功");
        }catch(UserNotFoundException e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }catch(PasswordWrongException e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
    }
    private static void login(String username, String password)
            throws UserNotFoundException, PasswordWrongException {
        if(!"admin".equals(username)) {
            //用户名不存在,对于用户而言要提示用户名或密码错误,但开发者需要知道真正的
            //原因
            throw new UserNotFoundException("用户名或密码错误!");
        }
        if(!"123456".equals(password)) {
            //密码错误
            throw new PasswordWrongException("用户名或密码错误!");
        }
    }
}
```

程序(代码清单 5.7)运行结果如图 5.6 所示。

```
请输入用户名：user
请输入密码：123
用户名或密码错误!
com.yyds.unit5.demo.UserNotFoundException Create breakpoint : 用户名或密码错误!
    at com.yyds.unit5.demo.Demo7Login.login(Demo7Login.java:29)
    at com.yyds.unit5.demo.Demo7Login.main(Demo7Login.java:13)
请输入用户名：admin
请输入密码：123
用户名或密码错误!
com.yyds.unit5.demo.PasswordWrongException Create breakpoint : 用户名或密码错误!
    at com.yyds.unit5.demo.Demo7Login.login(Demo7Login.java:33)
    at com.yyds.unit5.demo.Demo7Login.main(Demo7Login.java:13)
请输入用户名：admin
请输入密码：123456
登录成功
```

图 5.6　程序运行结果


扫一扫

5.3.2　方法重写中的异常

当一个类的方法声明了一个编译时异常后，它的子类如果重写该方法，重写方法声明的异常不能超过父类的异常，这里只遵循如下两点即可。运行时异常不受这两点约束。

（1）父类方法没有声明异常，子类重写该方法不能声明异常，如下所示。

```
class Parent {
    public void method1() {}
}
class Child extends Parent {
    //编译错误
    @Override
    public void method1() throws Exception {}
}
```

（2）父类方法声明了异常，子类重写该方法可以不声明异常，或者只声明父类的异常或该异常的子类，如下所示。

```
class Parent {
    public void method1() throws IOException {}
}
class Child extends Parent {
    //编译错误
    @Override
    public void method1() throws FileNotFoundException {}
}
```