Java 异常基础 Exception

Java中的异常 Exception

java.lang.Exception类是Java中所有异常的直接或间接父类。即Exception类是所有异常的根类。

比如程序：

```
public class ExceptionTest
{
	public static void main(String[] args)
	{
		int a = 3;
		int b = 0;
		int c = a / b;
		System.out.println(c);
	}
}
```

编译通过，执行时结果：

Exception in thread "main" java.lang.ArithmeticException: / by zero

at com.learnjava.exception.ExceptionTest.main(ExceptionTest.java:9)

因为除数为0，所以引发了算数异常。

比较常见的异常还有这种：空指针异常

java.lang.NullPointerException是空指针异常，出现该异常的原因在于某个引用为null，但却调用了它的某个方法，这时就会出现该异常。

Java中的异常分为两大类：

1.**Checked Exception**（非**Runtime Exception**）

2.**Unchecked Exception**（**Runtime Exception**）

运行时异常

RuntimeException类是Exception类的子类，它叫做运行时异常，Java中的所有运行时异常都会直接或者间接地继承自RuntimeException类。

Java中凡是继承自Exception，而不继承自RuntimeException类的异常都是非运行时异常。

异常处理的一般结构

```
        try
        {
             // 可能发生异常的代码
             // 如果发生了异常，那么异常之后的代码都不会被执行
        }
        catch (Exception e)
        {
            // 异常处理代码
        }
        finally
        {
            // 不管有没有发生异常，finally语句块都会被执行
        }
```

比如本文最开始的除法运算代码，加入异常处理之后：

```
    public class ExceptionTest
    {
        public static void main(String[] args)
        {
            int c = 0;
            try
```

```
        {
            int a = 3;
            int b = 0;

            // 这块代码出现了异常
            c = a / b;

            // 那么异常之后的代码都不会被执行
            System.out.println("Hello World");
        }
        catch (ArithmeticException e)
        {
            e.printStackTrace();
        }
        finally
        {
            //不管有没有发生异常，finally语句块都会被执行
            System.out.println("Welcome");
        }

        System.out.println(c);
        // 当b为0时，有异常，输出为c的初始值0
    }
}
```

多个**catch**

一个try后面可以跟多个catch，但不管多少个，最多只会有一个**catch**块被执行。

异常处理方法

对于非运行时异常（**checked exception**），必须要对其进行处理，否则无法通过编译。

处理方式有两种：

1.使用try..catch..finally进行捕获；

2.在产生异常的方法声明后面写上**throws** 某一个**Exception**类型，如throws Exception，将异常抛出到外面一层去。

对非运行时异常的处理详见代码例子：

处理方式**1**：将异常捕获

```
public class ExceptionTest2
{
    public void method() throws Exception // 将异常抛出，由调用这个方法的方法去处理这个异常,如果main方法也将异常抛出，则交给Java虚拟机来处理
    {
        System.out.println("Hello World");

        // 抛出异常
        throw new Exception();
    }

    public static void main(String[] args)
    {
        ExceptionTest2 test = new ExceptionTest2();

        try
        {
            test.method();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            System.out.println("Welcome");
        }


    }

}
```

处理方式**2**：将异常继续向外抛出

```
public class ExceptionTest2
{
    public void method() throws Exception // 将异常抛出，由调用这个方法的方法去处理这个异常,如果main方法也将异常抛出，则交给Java虚拟机来处理
    {
        System.out.println("Hello World");

        // 抛出异常
        throw new Exception();
    }
```

5. 观察者模式及Java实现例子(8)

```
    public static void main(String[] args) throws
Exception // main方法选择将异常继续抛出
    {
        ExceptionTest2 test = new ExceptionTest2();

        test.method(); // main方法需要对异常进行处理

        // 执行结果:
        // Hello World
        // Exception in thread "main"
java.lang.Exception
        // at
com.learnjava.exception.ExceptionTest2.method(Excep
tionTest2.java:10)
        // at
com.learnjava.exception.ExceptionTest2.main(Excepti
onTest2.java:17)
    }

}
```

对于运行时异常（**runtime exception**），可以对其进行处理，也可以不处理。推荐不对运行时异常进行处理。

自定义异常

所谓自定义异常，通常就是定义一个类，去继承Exception类或者它的子类。因为异常必须直接或者间接地继承自Exception类。

通常情况下，会直接继承自Exception类，一般不会继承某个运行时的异常类。

自定义异常可以用于处理用户登录错误，用户输入错误提示等。

自定义异常的例子：

自定义一个异常类型：

```
public class MyException extends Exception
{
    public MyException()
    {
```

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

一种异常处理方式：

```
public class ExceptionTest4
{

    public void method(String str) throws
MyException
    {
        if(null == str)
        {
            throw new MyException("传入的字符串参数不
能为null! ");
        }
        else
        {
            System.out.println(str);
        }
    }

    public static void main(String[] args) throws
MyException //异常处理方式1，不断向外抛出
    {
        ExceptionTest4 test = new ExceptionTest4();
        test.method(null);
    }
}
```

另一种异常处理方式：

```
public class ExceptionTest4
{

    public void method(String str) throws
MyException
    {
        if (null == str)
        {
            throw new MyException("传入的字符串参数不
能为null! ");
```

```
            }
            else
            {
                System.out.println(str);
            }
        }

        public static void main(String[] args)
        {
            //异常处理方式2，采用try...catch语句
            try
            {
                ExceptionTest4 test = new
    ExceptionTest4();
                test.method(null);

            }
            catch (MyException e)
            {
                e.printStackTrace();
            }
            finally
            {
                System.out.println("程序处理完毕");
            }

        }
    }
```

前面说过，可以有多个**catch**块，去捕获不同的异常，真正执行的时候最多只进入一个**catch**块。

下面这个例子，定义了两种自定义的异常类型：

```
    public class MyException extends Exception
    {

        public MyException()
        {
            super();
        }

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


    public class MyException2 extends Exception
```

```
{
    public MyException2()
    {
        super();
    }
    public MyException2(String message)
    {
        super(message);
    }

}


public class ExceptionTest4
{

    public void method(String str) throws
MyException, MyException2
    {
        if (null == str)
        {
            throw new MyException("传入的字符串参数不
能为null! ");
        }
        else if ("hello".equals(str))
        {
            throw new MyException2("传入的字符串不能
为hello");
        }
        else
        {
            System.out.println(str);
        }
    }

    public static void main(String[] args)
    {
        // 异常处理方式2，采用try...catch语句
        try
        {
            ExceptionTest4 test = new
ExceptionTest4();
            test.method(null);

        }
        catch (MyException e)
        {
            System.out.println("进入到MyException
catch块");
            e.printStackTrace();
        }
        catch (MyException2 e)
        {
            System.out.println("进入到MyException2
catch块");
```

```
                e.printStackTrace();
            }
            finally
            {
                System.out.println("程序处理完毕");
            }

        }
    }
```

我们可以使用多个catch块来捕获异常，这时需要将父类型的**catch**块放到子类型的**catch**块之后，这样才能保证后续的catch块可能被执行，否则子类型的catch块将永远无法到达，Java编译器会报错。

如果异常类型是独立的，那么它们的前后顺序没有要求。

如对上面的代码进行改动后，如下列出：

```
public class ExceptionTest4
{

    public void method(String str) throws Exception
// 也可以声明Exception，只要声明的可以涵盖所有抛出的异常即可
    {
        if (null == str)
        {
            throw new MyException("传入的字符串参数不能为null! ");
        }
        else if ("hello".equals(str))
        {
            throw new MyException2("传入的字符串不能为hello");
        }
        else
        {
            System.out.println(str);
        }
    }

    public static void main(String[] args)
    {
        // 异常处理方式2，采用try...catch语句
        try
```

```
            {
                ExceptionTest4 test = new
    ExceptionTest4();
                test.method(null);

            }
            catch (MyException e)
            {
                System.out.println("进入到MyException
    catch块");
                e.printStackTrace();
            }
            catch (MyException2 e)
            {
                System.out.println("进入到MyException2
    catch块");
                e.printStackTrace();
            }
            catch (Exception e)
            {
                //虽然需要加上，但是这块代码不会被执行，只是
    为了编译成功
                System.out.println("进入到MyException
    catch块");
                e.printStackTrace();
                //如果去掉前面两个catch块或其中之一，则发生
    该异常时就会进入此catch块
                //catch块的匹配是按照从上到下的顺序，所以这
    个块如果放在最前面就会捕获所有的异常，后面的块永远不会执
    行，这时候会提示编译错误
            }
            finally
            {
                System.out.println("程序处理完毕");
            }

        }
    }
```

面试常考题型

try块中的退出语句

虽然实际开发中不会遇到这样的情况，但是笔试面试时有关异常经常会问到如下情况：

```
    public class ExceptionTest5
    {
```

```
    public void method()
    {
        try
        {
            System.out.println("进入到try块");

            //return;
            //会先执行finally块再返回

            //虚拟机退出
            //System.exit(0);
            //不会执行finally块中的语句，直接退出
        }
        catch (Exception e)
        {
            System.out.println("异常发生了！ ");

        }
        finally
        {
            System.out.println("进入到finally块");

        }

        System.out.println("后续代码");

    }

    public static void main(String[] args)
    {
        ExceptionTest5 test = new ExceptionTest5();
        test.method();
    }
}
```

在加上return语句前，程序输出：

进入到try块

进入到finally块

后续代码

如果在try块中加入return语句：

程序执行输出：

进入到try块

进入到finally块

说明**try**块中有**return**语句时，仍然会首先执行**finally**块中的语句，然后方法再返回。

如果try块中存在System.exit(0);语句，那么就不会执行finally块中的代码，因为System.exit(0)会终止当前运行的Java虚拟机，程序会在虚拟机终止前结束执行。

参考资料

圣思园张龙老师Java SE系列视频教程。

分类: Java

标签: Java, 笔试面试

« 上一篇：结合JUnit来说明反射和注解的用途
» 下一篇：Java AWT基础及布局管理

posted on 2013-02-03 16:12 圣骑士wind 阅读(28252) 评论(8) 编辑 收藏

评论

#1楼

写的很详细，优雅地赞一个

支持(0) 反对(0)

2014-10-08 20:42 | fomeiherz

#2楼

很有条理，赞。

支持(0) 反对(0)

2015-01-13 15:09 | 笺陌

#3楼

顶

支持(0) 反对(0)

2015-03-25 23:14 | Zempty

#4楼

看过很多关于异常的文章，相比较而言，这篇文章讲解的更深入更详细：异常的深入研究与分析

支持(0) 反对(0)

2015-05-30 22:39 | netwelfare

#5楼

可以的，刚学java，为了关注，注册了个用户

支持(0) 反对(0)

2015-10-16 16:32 | 永远爱妮

#6楼

好！

支持(0) 反对(0)

2015-12-14 16:20 | sunshine小北

#7楼

不错，赞一个。

支持(0) 反对(0)

2016-01-29 11:21 | 笑吧

#8楼

张老师思路很清晰

支持(0) 反对(0)

2016-03-05 06:36 | 归 |

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新**IT**新闻:

· 谷歌地图：当年东日本大地震造成的破坏如今是怎样一番景象
· 微软向Linux系统开放SQL数据库软件
· 寨卡病毒为何可怕？它能感染杀死大脑干细胞
· 智能手表遇冷？Apple Watch 2可不这么想
· 波兰科学家称"新物理学"黎明或到来

» 更多新闻...

最新知识库文章:

· 谷歌背后的数学
· Medium开发团队谈架构设计
· 理解"渐进增强(Progressive Enhancement)"
· 为什么说DOM操作很慢
· 为什么你应该尝试全栈

» 更多知识库文章...