首页 Java基础 Web开发 FreeMarker Json与Bean Eclipse大师 字符和编码 资源下载 关于网站 搜索

Java基础 异常的深入研究与分析(1)

异常的深入研究与分析(1)

栏目: Java基础 作者: admin 日期: 2015-05-30 评论: 0 点击: 3,381 次

前言

本文是异常内容的集大成者,力求全面,深入的异常知识研究与分析。本文由金丝燕网独家撰写,参考众多网上资源,经过内容辨别取舍,文字格式校验等步骤编辑而成,以飨读者。对于本文的内容,建议小白需要多多思考力求掌握,对于老手只需意会温故知新。对于本文的内容,属于基础知识研究范畴,切勿以为读完此文就能将异常知识掌握到家。切记:操于曲而后晓声,观于剑而后识器,所以我觉得没有大量的源码阅读经验,你很难知道什么时候需要自定义异常,什么时候需要抛出异常。

异常机制概述

异常机制是指当程序出现错误后,程序如何处理。具体来说,异常机制提供了程序退出的安全通道。当出现错误后,程序执行的流程发生改变,程序的控制权转移到异常处理器。

异常处理的流程

当程序中抛出一个异常后,程序从程序中导致异常的代码处跳出,java虚拟机检测寻找和try关键字匹配的处理该异常的catch块,如果找到,将控制权交到catch块中的代码,然后继续往下执行程序,try块中发生异常的代码不会被重新执行。如果没有找到处理该异常的catch块,在所有的finally块代码被执行和当前线程的所属的ThreadGroup的uncaughtException方法被调用后,遇到异常的当前线程被中止。

异常的结构

异常的继承结构: Throwable为基类, Error和Exception继承Throwable, RuntimeException和IOException等继承Exception。Error和RuntimeException及其子类成为未检查异常(unchecked), 其它异常成为已检查异常(checked)。

最新文章

关于域名的若干想法

公司的伙食

基于人工智能的问答类型APP介绍

来年寄语

经验决定行动

主机迁移公告

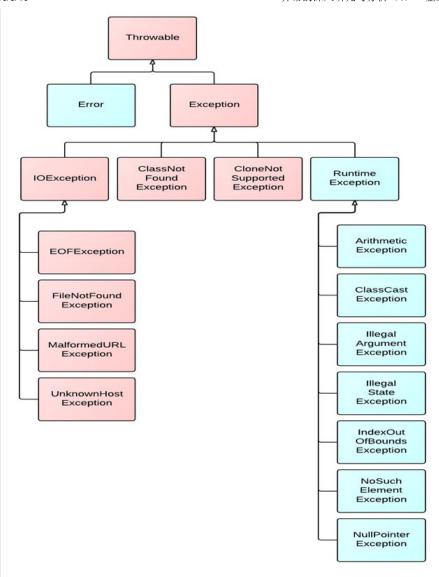
UTF-16编码和Java, UTF-8编码...

UTF-8编码

Big Endian和Little Endian

UTF-16编码

http://swiftlet.net/archives/998



Error异常

Error表示程序在运行期间出现了十分严重、不可恢复的错误,在这种情况下应用程序只能中止运行,例如JAVA 虚拟机出现错误。Error是一种unchecked Exception,编译器不会检查Error是否被处理,在程序中不用捕获Error类型的异常。一般情况下,在程序中也不应该抛出Error类型的异常。

RuntimeException异常

Exception异常包括RuntimeException异常和其他非RuntimeException的异常。

RuntimeException 是一种Unchecked Exception,即表示编译器不会检查程序是否对RuntimeException作了处理,在程序中不必捕获RuntimException类型的异常,也不必在方法体声明抛出RuntimeException类。RuntimeException发生的时候,表示程序中出现了编程错误,所以应该找出错误修改程序,而不是去捕获RuntimeException。

Checked Exception异常

Checked Exception异常,这也是在编程中使用最多的Exception,所有继承自Exception并且不是RuntimeException的异常都是checked Exception,上图中的IOException和ClassNotFoundException。JAVA 语言规定必须对checked Exception作处理,编译器会对此作检查,要么在方法体中声明抛出checked Exception,要么使用catch语句捕获checked Exception进行处理,不然不能通过编译。

在声明方法时候抛出异常

语法:throws(略)

为什么要在声明方法抛出异常?

方法是否抛出异常与方法返回值的类型一样重要。假设方法抛出异常却没有声明该方法将抛出异常,那么客户程序员可以调用这个方法而且不用编写处理异常的代码。那么,一旦出现异常,那么这个异常就没有合适的异常控制器来解决。 为什么抛出的异常一定是已检查异常?

RuntimeException与Error可以在任何代码中产生,它们不需要由程序员显示的抛出,一旦出现错误,那么相应的异常会被自动抛出。遇到Error,程序员一般是无能为力的;遇到RuntimeException,那么一定是程序存在逻辑错误,要对程序进行修改;只有已检查异常才是程序员所关心的,程序应该且仅应该抛出或处理已检查异常。而已检查异常是由程序员抛出的,这分为两种情况:客户程序员调用会抛出异常的库函数;客户程序员自己使用throw语句抛出异常。

http://swiftlet.net/archives/998

注意:

覆盖父类某方法的子类方法不能抛出比父类方法更多的异常,所以,有时设计父类的方法时会声明抛出异常,但实际的实现方法的代码却并不抛出异常,这样做的目的就是为了方便子类方法覆盖父类方法时可以抛出异常。

在方法中如何抛出异常

语法:throw(略)

抛出什么异常?

对于一个异常对象,真正有用的信息是异常的对象类型,而异常对象本身毫无意义。比如一个异常对象的类型是 ClassCastException,那么这个类名就是唯一有用的信息。所以,在选择抛出什么异常时,最关键的就是选择异常的类 名能够明确说明异常情况的类。

异常对象通常有两种构造函数:一种是无参数的构造函数;另一种是带一个字符串的构造函数,这个字符串将作为这个异常对象除了类型名以外的额外说明。

```
为什么要创建自己的异常?
```

当Java内置的异常都不能明确的说明异常情况的时候,需要创建自己的异常。需要注意的是,唯一有用的就是类型名这个信息,所以不要在异常类的设计上花费精力。

throw和throws的区别

```
public class TestThrow
2
      public static void main(String[] args)
4
5
6
7
          {
              //调用带throws声明的方法,必须显式捕获该异常
                  则,必须在main方法中再次声明抛出
              throwChecked(-3);
10
          catch (Exception e)
11
12
13
              System.out.println(e.getMessage());
14
          //调用抛出Runtime异常的方法既可以显式捕获该异常,
15
16
17
          throwRuntime(3);
18
19
      public static void throwChecked(int a)throws Exception
20
21
          if (a > 0)
22
          {
23
              //自行抛出Exception异常
24
              //该代码必须处于try块里,或处于带throws声明的方法中
25
              throw new Exception("a的值大于0,不符合要求");
26
          }
27
28
      public static void throwRuntime(int a)
29
30
          if (a > 0)
31
          {
32
              //自行抛出RuntimeException异常,既可以显式捕获该异常
33
34
              throw new RuntimeException("a的值大于0,不符合要求");
          }
35
36
      }
37
```

补充:throwChecked函数的另外一种写法如下所示:

```
public static void throwChecked(int a)
      {
3
           if (a > 0)
4
               //自行抛出Exception异常
5
6
7
               //该代码必须处于try块里,或处于带throws声明的方法中
               try
              {
                  throw new Exception("a的值大于0,不符合要求");
              }
10
11
              catch (Exception e)
12
13
                    / TODO Auto-generated catch block
14
                  e.printStackTrace();
15
              }
          }
16
17
```

注意:此时在main函数里面throwChecked就不用try异常了。

应该在声明方法抛出异常还是在方法中捕获异常?

处理原则:捕捉并处理哪些知道如何处理的异常,而传递哪些不知道如何处理的异常

使用finally块释放资源

 $finally关键字保证无论程序使用任何方式离开try块\ ,\ finally中的语句都会被执行。在以下三种情况下会进入finally块:$

http://swiftlet.net/archives/998

- (1) try块中的代码正常执行完毕。
- (2) 在try块中抛出异常。
- (3) 在try块中执行return、break、continue。

因此,当你需要一个地方来执行在任何情况下都必须执行的代码时,就可以将这些代码放入finally块中。当你的程序中使用了外界资源,如数据库连接,文件等,必须将释放这些资源的代码写入finally块中。

必须注意的是:在finally块中不能抛出异常。JAVA异常处理机制保证无论在任何情况下必须先执行finally块然后再离开 try块,因此在try块中发生异常的时候,JAVA虚拟机先转到finally块执行finally块中的代码,finally块执行完毕后,再 向外抛出异常。如果在finally块中抛出异常,try块捕捉的异常就不能抛出,外部捕捉到的异常就是finally块中的异常信息,而try块中发生的真正的异常堆栈信息则丢失了。

请看下面的代码:

```
Connection con = null;
   try
3
4
   {
       con = dataSource.getConnection();
6
7
   catch(SQLException e)
10
       throw e;//进行一些处理后再将数据库异常抛出给调用者处理
   finally
12
13
14
       try
15
16
           con.close();
17
       catch(SQLException e)
18
19
  {
20
       e.printStackTrace();
21
22
23
```

运行程序后,调用者得到的信息如下

java.lang.NullPointerException

at myPackage.MyClass.method1(methodl.java:266)

而不是我们期望得到的数据库异常。这是因为这里的con是null的关系,在finally语句中抛出了NullPointerException,在finally块中增加对con是否为null的判断可以避免产生这种情况。

丢失的异常

请看下面的代码:

```
public void method2()
   try
4
       method1(); //method1进行了数据库操作
6
7
   catch(SQLException e)
9
10
       throw new MyException("发生了数据库异常:"+e.getMessage);
11
12
13
14
   public void method3()
15
16
       try
17 {
18
       method2();
19 }
   catch(MyException e)
20
21
22
       e.printStackTrace();
23
24
25
```

上面method2的代码中,try块捕获method1抛出的数据库异常SQLException后,抛出了新的自定义异常MyException。这段代码是否并没有什么问题,但看一下控制台的输出:

MyException:发生了数据库异常:对象名称'MyTable' 无效。

at MyClass.method2(MyClass.java:232)

at MyClass.method3(MyClass.java:255)

原始异常SQLException的信息丢失了,这里只能看到method2里面定义的MyException的堆栈情况;而method1中发生的数据库异常的堆栈则看不到,如何排错呢,只有在method1的代码行中一行行去寻找数据库操作语句了。JDK的开发者们也意识到了这个情况,在JDK1.4.1中,Throwable类增加了两个构造方法,public Throwable(Throwable cause)和public Throwable(String message,Throwable cause),在构造函数中传入的原始异常堆栈信息将会在printStackTrace方法中打印出来。但对于还在使用JDK1.3的程序员,就只能自己实现打印原始异常堆栈信息的功能了。实现过程也很简单,只需要在自定义的异常类中增加一个原始异常字段,在构造函数中传入原始异常,然后重载printStackTrace方法,首先调用类中保存的原始异常的printStackTrace方法,然后再调用super.printStackTrace方法就可以打印出原始异常信息了。可以这样定义前面代码中出现的MyException类:

http://swiftlet.net/archives/998 4/5

import jow.io.PrintStream;	2 imp 3 pub 4 { 5 6	
3 public class MyException extends Exception 4 { 5	3 pub 4 { 5 6	
# f	4 { 5 6	
for private static final long serialVersionUID = 11; // // // // // // // // // // // // //	5	tte class myexception excends exception
7 / 於與音樂 private Throwable cause; 2 / / 地面演戲 public MyException(Throwable cause) {		
### Private Throwable cause;		
public MyException(Throwable cause) { this.cause = cause; } public MyException(String s, Throwable cause) { this.cause = cause; } public MyException(String s, Throwable cause) { this.cause = cause; } // ExprintStackTrace(); } public void printStackTrace(); } public void printStackTrace(); } public void printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(PrintStream s) } if (cause != null) { cause.printStackTrace(s); } public void printStackTrace(s); if (cause != null) { cause.printStackTrace(s); } public void printStackTrace(s); } public void printStackTrace(s); public void printStackTrace(s); } public void printStackTrace(s); if (cause != null) { cause.printStackTrace(s); } public void printStackTrace(s); } public void printStackTrace(s); public void printStack		
# public MyException(Throwable cause) {		
{ this.cause = cause; } public MyException(String s, Throwable cause) {		
public MyException(String s,Throwable cause) { super(s); this.cause = cause; } // # RyprintStackTrace/ik. 打印机场外介地模型 public void printStackTrace(); } super.printStackTrace(); } super.printStackTrace(); } super.printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(PrintStream s) if (cause != null) { cause.printStackTrace(PrintWriter s) } } super.printStackTrace(s); } super.printStackTrace(PrintWriter s) { if (cause != null) { cause.printStackTrace(PrintWriter s) } } if (cause != null) { cause.printStackTrace(S); } super.printStackTrace(S); } ### ### ### ### ### ### ###		
# public MyException(String s,Throwable cause) {		
super(s); this.cause = cause; } // zwprintStackTrace/ik. 打印出版前异常接接总 public void printStackTrace(); } super.printStackTrace(); } super.printStackTrace(); } super.printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(PrintStream s) } if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(printWriter s) { if (cause != null) } cause.printStackTrace(printWriter s) } if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); } super.printStackTrace(s); } super.printStackTrace(s); } super.printStackTrace(s); #### ###############################		
super(s); this.cause = cause; } //### public void printStackTrace() {		
/## WprintStackTrace/》: 打印出版始异常集核信息 public void printStackTrace() { if (cause != null)		
9 /		
public void printStackTrace() { if (cause!= null) cause.printStackTrace(); } super.printStackTrace(); } public void printStackTrace(PrintStream s) { if (cause!= null) { cause.printStackTrace(PrintWriter s) } } public void printStackTrace(S); } public void printStackTrace(S); fi (cause!= null) { cause.printStackTrace(PrintWriter s) } if (cause!= null) { cause.printStackTrace(s); } public void printStackTrace(S); public void printStackTrace(PrintWriter s) public vo		
{ cause.printStackTrace(); } super.printStackTrace(); } super.printStackTrace(); } public void printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); } super.printStackTrace(PrintWriter s) { if (cause != null) { cause.printStackTrace(PrintWriter s) { if (cause != null) { cause.printStackTrace(s); } } super.printStackTrace(s); } super.		
ause.printStackTrace(); } super.printStackTrace(); } public void printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(s); } public void printStackTrace(s); } public void printStackTrace(s); fif (cause != null) { cause.printStackTrace(PrintWriter s) { if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); ### A **The **		if (cause != null)
\$ super.printStackTrace(); } public void printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(s); } public void printStackTrace(s); } public void printStackTrace(s); f (cause != null) { cause.printStackTrace(PrintWriter s) } if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); #### was a super.printStackTrace(s); ###################################		
super.printStackTrace(); public void printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); } public void printStackTrace(PrintWriter s) { if (cause != null) { cause.printStackTrace(PrintWriter s) } if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); } super.printStackTrace(s); } super.printStackTrace(s); ## supe		
### public void printStackTrace(PrintStream s) if (cause != null)		
public void printStackTrace(PrintStream s) { if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); } public void printStackTrace(PrintWriter s) if (cause != null) { cause.printStackTrace(PrintWriter s) } if (cause != null) { cause.printStackTrace(s); } super.printStackTrace(s); } im: avalae丝燕网原创编译,转载请保留链接: 异常的深入研究与分析(1) im: javal j派研究分析报告 im: 异常的深入研究与分析(2) *** ** ** ** ** ** ** ** **		
f		
if (cause != null) {		
Cause.printStackTrace(s);		
Cause.printStackTrace(s);		
super.printStackTrace(s); } public void printStackTrace(PrintWriter s) { if (cause != null) { cause.printStackTrace(s); } } super.printStackTrace(s); } super.printStackTrace(s); } super.printStackTrace(s); super.printStackTrace(printWriter s) super.printStackTrace(s); super.printStac		<pre>cause.printStackTrace(s);</pre>
public void printStackTrace(PrintWriter s)		
public void printStackTrace(PrintWriter s) { if (cause != null) { cause.printStackTrace(s); } ### A 文由金丝燕网原创编译,转载清保留链接:异常的深入研究与分析(1) ###################################		
1		j
### (cause != null) {		<pre>public void printStackTrace(PrintWriter s)</pre>
12		
22		
3		
15 } \$	13	
書明: 本文由金丝燕网原创编译,转载请保留链接: 异常的深入研究与分析(1) 篇: Java门派研究分析报告篇: 异常的深入研究与分析(2) **常的深入研究与分析(1): 等您坐沙发呢! ***********************************		
唐明:本文由金丝燕网原创编译,转载请保留链接:异常的深入研究与分析(1) —篇:Java门派研究分析报告 —篇:异常的深入研究与分析(2) *常的深入研究与分析(1):等您坐沙发呢! 发表评论		}
上一篇: Java门派研究分析报告 下一篇: 异常的深入研究与分析 (2) 整常的深入研究与分析 (1): 等您坐沙发呢! 发表评论 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□		
—篇: Java门派研究分析报告 —篇: 异常的深入研究与分析 (2) #常的深入研究与分析 (1): 等您坐沙发呢! 发表评论 □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □	±nn →	
完一篇: 异常的深入研究与分析(1): 等您坐沙发呢! 发表评论 明称 * 邮箱 * 网址 (2) (2) (3) (3) (3) (3) (4) (9) (9) (9) (9) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1		
□ 完善: 异常的深入研究与分析(1): 等您坐沙发呢! 发表评论 □ 即称 * □ 即称 * □ 即位 *** ○ 回 **** ○ ***	当明: 本	· 文田並並然例於的無洋,特敦俱休田始安· 并带的水入则先一刀机(I)
完一篇:异常的深入研究与分析(1):等您坐沙发呢! 发表评论	=明: ⊄	· 人口亚丝然例成切别样,轻软相体由键线· 并带切床入则九一刀和(I)
#常的深入研究与分析 (1):等您坐沙发呢! FR *	≕明:	· 文 田亚 丝 然(M)
发表评论 明称 * 明箱 *		
发表评论 明符 * 明符 *	篇:	Java门派研究分析报告
发表评论 ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・	篇:	Java门派研究分析报告
	篇:	Java门派研究分析报告 异常的深入研究与分析(2)
#箱 *	篇:	Java门派研究分析报告 异常的深入研究与分析(2)
邮箱 *	:─篇: -─篇: * 常的深	Java门派研究分析报告 异常的深入研究与分析(2) 入研究与分析(1):等您坐沙发呢!
	:一篇: -一篇: 常的深	Java门派研究分析报告 异常的深入研究与分析(2) 入研究与分析(1):等您坐沙发呢!
	:─篇: 	Java门派研究分析报告 异常的深入研究与分析(2) 入研究与分析(1):等您坐沙发呢!
~ \(\bullet \)	:─篇: -─篇: * 常的深	Java门派研究分析报告 异常的深入研究与分析(2) 入研究与分析(1):等您坐沙发呢! &
	:─篇: -─篇: * 常的深	Java门派研究分析报告 异常的深入研究与分析(2) 入研究与分析(1):等您坐沙发呢! &
提交[Ctrl+Enter] 重写	二一篇: 一篇: 学常的深。 发表评词	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter] 重写	二一篇: 一篇: 常的深 发表评i	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter]	二一篇: 一篇: 学常的深。 发表评词	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter] 重写	上一篇: 下一篇: 发表评 i	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter] 重写	上一篇: 下一篇: 发表评 i	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter]	上一篇: 下一篇: 发表评 i	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter] 重写	上一篇: 下一篇: 学常的深 发表评	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter] 重写	上一篇: 下一篇: 发表评 i	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
提交[Ctrl+Enter]	上一篇: 下一篇: 发表评 i	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
western such	上一篇: 下一篇: 学常的深 发表评	Java门派研究分析报告 异常的深入研究与分析 (2) 入研究与分析 (1):等您坐沙发呢! &
	上一篇: 下一篇: 学常的深 发表评i	Java门派研究分析报告 异常的深入研究与分析 (2)
	上一篇: 下一篇: 学常的深 发表评 i	Java门派研究分析报告 异常的深入研究与分析 (2)
	上一篇: 下一篇: 学常的深 发表评i	Java门派研究分析报告 异常的深入研究与分析 (2)

版权所有:金丝燕网 | 运行时间: 409天

免责声明:本站所有pdf书籍和教程视频均来自于互联网,由热心读者共同搜集,仅限于个人学习与研究,严禁用于商业用途。 原作者如果认为本站侵犯了您的版权,请及时告知,本站会立即删除! 金丝燕网站 Copyright (c) 2014 www.swiftlet.net All rights reserved



http://swiftlet.net/archives/998 5/5