

圣骑士Wind的博客

技术让生活更美好。

随笔 - 368, 文章 - 8, 评论 - 419, 引用 - 0

导航

博客园
首 页
新随笔
联 系
订 阅
管 理

XML

<	2013年2月						>
日	一	二	三	四	五	六	
27	28	29	30	31	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	1	2	
3	4	5	6	7	8	9	

公告

欢迎来到圣骑士**Wind**的博客！

友情推广技术博客：
WeYoung

Links:

github
Android API Guides
GrepCode Android

昵称：圣骑士wind

园龄：4年6个月

粉丝：799

关注：67

+加关注

搜索

<input type="text"/>	找找看
<input type="text"/>	谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

Android(116)
Java(83)

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。

Web(38)
JavaScript(29)
设计模式(29)
数据结构和算法(17)
JSP(15)
Servlet(14)
JUnit(13)
XML(12)
更多

随笔分类(432)

3D基础(3)
Android(90)
Android Animation(8)
Android Testing(3)
Android 进阶(7)
C#(3)
C++(1)
CMake(2)
DirectX(3)
Git(7)
GPU(1)
Java(81)
JavaScript(24)
JNI(1)
jQuery(10)
JUnit(13)
Linux(6)
MySQL(3)
OpenCV(2)
OpenGL(1)
OpenGL ES(1)
Python(2)
Testing(1)
UX(5)
Web(28)
Windows(2)
XML(12)
XNA(1)
笔试面试(2)
编程调试(4)
服务器(3)
科研什么的(12)
每天一个算法题(10)
软件工具(1)
设计模式(29)
数据结构和算法(19)
数据库基础(6)
图形图像(2)
图形学(10)
网络(13)

随笔档案(367)

2015年10月 (1)
2015年6月 (1)
2015年5月 (3)
2015年4月 (5)
2015年3月 (1)

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
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，加入异常处理后：

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

2015年2月 (1)
2015年1月 (6)
2014年12月 (4)
2014年9月 (2)
2014年7月 (3)
2014年6月 (1)
2014年5月 (4)
2014年4月 (17)
2014年3月 (10)
2014年2月 (11)
2014年1月 (3)
2013年12月 (4)
2013年11月 (7)
2013年10月 (2)
2013年9月 (11)
2013年8月 (14)
2013年7月 (10)
2013年6月 (34)
2013年5月 (26)
2013年4月 (21)
2013年3月 (38)
2013年2月 (25)
2013年1月 (38)
2012年12月 (18)
2012年11月 (20)
2012年10月 (4)
2012年9月 (3)
2012年8月 (13)
2012年7月 (2)
2012年5月 (1)
2012年4月 (2)
2011年8月 (1)

Android学习

Android源码查看
Android源码项目git
老罗的Android之旅

官方文档

Android 官网
Git Reference
Python
Python官网

开源社区

GitHub
Google Code

学习链接

A Byte of Python

积分与排名

积分 - 660088
排名 - 117

最新评论

1. Re:Java 异常基础
Exception
张老师思路很清晰

```
{
    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

--归 |

2. Re:Android SQLite数据库使用 学习与代码实践

@

Activity里面的

queryTheCursor有什么

用,而且getColumnName

这个方法怎么出来的?

--Wendell_Sir

3. Re:Android SQLite数据库使用 学习与代码实践

蛮好

--Wendell_Sir

4. Re:Git 分支管理和冲突解决

清楚,好文

--小沐风

5. Re:四元数基础

哥们, 四元数叉×公式, x,

y,z 三个分量的计算中, 正

负号搞错了。const float

w = (l.w * r.w) - (l.x * r.x)

- (l.y * r.y) - (l.....

--蔡志刚

阅读排行榜

1. Android Fragment 基本介绍(95700)

2. Android 抽屉效果的导航菜单实现(90720)

3. Android PopupWindow的使用和分析(89992)

4. Android Fragment和Activity(81353)

5. Android WebView使用基础(77819)

评论排行榜

1. Google Maps Android API V2的使用及问题解决(151)

2. OpenCV实现人脸检测(13)

3. Android HTTP实例 使用GET方法和POST方法发送请求(13)

4. Android 抽屉效果的导航菜单实现(12)

5. Java 多线程(七) 线程间的通信(10)

推荐排行榜

1. Android Fragment 基本介绍(14)

2. Java 多线程(六) synchronized关键字详解(13)

3. Google Maps Android API V2的使用及问题解决(11)

4. Android 抽屉效果的导航菜单实现(9)



```
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(Excepti
onTest2.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。

如对上述的代码进行改动后，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](#)

好文要顶

关注我

收藏该文



圣骑士wind

关注 - 67

粉丝 - 799

[+加关注](#)

5

0

(请您对文章做出评价)

« 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](#)。

【推荐】50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

【推荐】融云即时通讯云—专注为 App 开发者提供IM云服务

【推荐】UCloud开年大礼，充5000返1000；买云主机送CDN，[详情点击](#)



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...

Powered by:

博客园

Copyright © 圣骑士wind

