@来源 Java 异常处理的十个建议 (qq.com)

前言

Java异常处理的十个建议，希望对大家有帮助~

本文已上传github:

```
https://github.com/whx123/JavaHome
```

公众号：捡田螺的小男孩

一、尽量不要使用e.printStackTrace(),而是使用log打印。

反例:

```
try{
  // do what you want
}catch(Exception e){
  e.printStackTrace();
}
```

正例:

```
try{
  // do what you want
}catch(Exception e){
  log.info("你的程序有异常啦,{}",e);
}
```

理由:

- printStackTrace()打印出的堆栈日志跟业务代码日志是交错混合在一起的，排查异常日志不太方便。
- e.printStackTrace()语句产生的字符串记录的是堆栈信息，如果信息太长太多，字符串常量池所在的内存块没有空间了,即内存满了，那么，用户的请求就卡住啦~

二、catch了异常，但是没有打印出具体的exception，无法更好定位问题

反例:

```
try{
  // do what you want
}catch(Exception e){
  log.info("你的程序有异常啦");
}
```

正例:

```
try{
  // do what you want
}catch(Exception e){
  log.info("你的程序有异常啦，{}",e);
}
```

理由：

- 反例中，并没有把exception出来，到时候排查问题就不好查了啦，到底是SQl写错的异常还是IO异常，还是其他呢？所以应该把exception打印到日志中哦~

三、不要用一个Exception捕捉所有可能的异常

反例：

```
public void test(){
    try{
        //…抛出 IOException 的代码调用
        //…抛出 SQLException 的代码调用
    }catch(Exception e){
        //用基类 Exception 捕捉的所有可能的异常，如果多个层次都这样捕捉，会丢失原始异常的有效信息哦
        log.info(“Exception in test,exception:{}”, e);
    }
}
```

正例：

```
public void test(){
    try{
        //…抛出 IOException 的代码调用
        //…抛出 SQLException 的代码调用
    }catch(IOException e){
        //仅仅捕捉 IOException
        log.info(“IOException in test,exception:{}”, e);
    }catch(SQLException e){
        //仅仅捕捉 SQLException
        log.info(“SQLException in test,exception:{}”, e);
    }
}
```

理由：

- 用基类 Exception 捕捉的所有可能的异常，如果多个层次都这样捕捉，会丢失原始异常的有效信息哦

四、记得使用finally关闭流资源或者直接使用try-with-resource

反例：

```
FileInputStream fdIn = null;
try {
    fdIn = new FileInputStream(new File("/jay.txt"));
    //在这里关闭流资源？有没有问题呢？如果发生异常了呢？
    fdIn.close();
} catch (FileNotFoundException e) {
    log.error(e);
} catch (IOException e) {
    log.error(e);
}
```

正例1：

需要使用finally关闭流资源，如下

```
FileInputStream fdIn = null;
try {
    fdIn = new FileInputStream(new File("/jay.txt"));
} catch (FileNotFoundException e) {
    log.error(e);
} catch (IOException e) {
    log.error(e);
}finally {
    try {
        if (fdIn != null) {
            fdIn.close();
        }
    } catch (IOException e) {
        log.error(e);
    }
}
```

正例2：

当然，也可以使用JDK7的新特性try-with-resource来处理，它是Java7提供的一个新功能，它用于自动资源管理。

- 资源是指在程序用完了之后必须要关闭的对象。
- try-with-resources保证了每个声明了的资源在语句结束的时候会被关闭
- 什么样的对象才能当做资源使用呢？只要实现了java.lang.AutoCloseable接口或者java.io.Closeable接口的对象，都OK。

```
try (FileInputStream inputStream = new FileInputStream(new File("jay.txt")) {
    // use resources
} catch (FileNotFoundException e) {
    log.error(e);
} catch (IOException e) {
    log.error(e);
}
```

理由：

- 如果不使用finally或者try-with-resource，当程序发生异常，IO资源流没关闭，那么这个IO资源就会被他一直占着，这样别人就没有办法用了，这就造成资源浪费。

五、捕获异常与抛出异常必须是完全匹配，或者捕获异常是抛异常的父类

反例：

```
//BizException 是 Exception 的子类
public class BizException extends Exception {}
//抛出父类Exception
public static void test() throws Exception {}

try {
    test(); //编译错误
} catch (BizException e) { //捕获异常子类是没法匹配的哦
    log.error(e);
}
```

正例：

```
//抛出子类Exception
public static void test() throws BizException {}

try {
    test();
} catch (Exception e) {
    log.error(e);
}
```

六、捕获到的异常，不能忽略它，至少打点日志吧

反例：

```
public static void testIgnoreException() throws Exception {
    try {
        // 搞事情
    } catch (Exception e) {     //一般不会有这个异常

    }
}
```

正例：

```
public static void testIgnoreException() {
    try {
        // 搞事情
    } catch (Exception e) {     //一般不会有这个异常
        log.error("这个异常不应该在这里出现的,{}",e);
    }
}
```

理由：

- 虽然一个正常情况都不会发生的异常，但是如果你捕获到它，就不要忽略呀，至少打个日志吧~

七、注意异常对你的代码层次结构的侵染（早发现早处理）

反例：

```
public UserInfo queryUserInfoByUserId(Long userid) throw SQLException {
    //根据用户Id查询数据库
}
```

正例：

```
public UserInfo queryUserInfoByUserId(Long userid) {
    try{
        //根据用户Id查询数据库
    }catch(SQLException e){
        log.error("查询数据库异常啦，{}",e);
    }finally{
        //关闭连接，清理资源
    }
}
```

理由：

- 我们的项目，一般都会把代码分 Action、Service、Dao 等不同的层次结构，如果你是DAO层处理的异常，尽早处理吧，如果往上 throw SQLException，上层代码就还是要try catch处理啦，这就污染了你的代码~

八、自定义封装异常，不要丢弃原始异常的信息Throwable cause

我们常常会想要在捕获一个异常后抛出另一个异常，并且希望把原始异常的信息保存下来，这被称为异常链。公司的框架提供统一异常处理就用到异常链，我们自定义封装异常，不要丢弃原始异常的信息，否则排查问题就头疼啦

反例：

```
public class TestChainException {
    public void readFile() throws MyException{
        try {
            InputStream is = new FileInputStream("jay.txt");
            Scanner in = new Scanner(is);
            while (in.hasNext()) {
                System.out.println(in.next());
            }
        } catch (FileNotFoundException e) {
            //e 保存异常信息
            throw new MyException("文件在哪里呢");
        }
    }
    public void invokeReadFile() throws MyException{
        try {
            readFile();
        } catch (MyException e) {
            //e 保存异常信息
            throw new MyException("文件找不到");
        }
    }
    public static void main(String[] args) {
```

```
        TestChainException t = new TestChainException();
        try {
            t.invokeReadFile();
        } catch (MyException e) {
            e.printStackTrace();
        }
    }
}
//MyException 构造器
public MyException(String message) {
        super(message);
}
```

运行结果如下，没有了Throwable cause，不好排查是什么异常了啦

```
exception.MyException
    at exception.TestChainException.invokeReadFile(TestChainException.java:28)
    at exception.TestChainException.main(TestChainException.java:35)
Caused by: exception.MyException: 文件在哪里呢
    at exception.TestChainException.readFile(TestChainException.java:19)
    at exception.TestChainException.invokeReadFile(TestChainException.java:25)
    ... 1 more
```


正例：

```
public class TestChainException {
    public void readFile() throws MyException{
        try {
            InputStream is = new FileInputStream("jay.txt");
            Scanner in = new Scanner(is);
            while (in.hasNext()) {
                System.out.println(in.next());
            }
        } catch (FileNotFoundException e) {
            //e 保存异常信息
            throw new MyException("文件在哪里呢");
        }
    }
    public void invokeReadFile() throws MyException{
        try {
            readFile();
        } catch (MyException e) {
            //e 保存异常信息
            throw new MyException("文件找不到");
        }
    }
    public static void main(String[] args) {
        TestChainException t = new TestChainException();
        try {
            t.invokeReadFile();
        } catch (MyException e) {
            e.printStackTrace();
        }
    }
}
```

```
//MyException 构造器
public MyException(String message) {
        super(message);
}
```

```
exception.MyException
    at exception.TestChainException.invokeReadFile(TestChainException.java:28)
    at exception.TestChainException.main(TestChainException.java:35)
Caused by: exception.MyException
    at exception.TestChainException.readFile(TestChainException.java:19)
    at exception.TestChainException.invokeReadFile(TestChainException.java:25)
    ... 1 more
Caused by: java.io.FileNotFoundException: jay.txt (系统找不到指定的文件。)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.io.FileInputStream.<init>(FileInputStream.java:93)
    at exception.TestChainException.readFile(TestChainException.java:13)
    ... 2 more
```


九、运行时异常RuntimeException，不应该通过catch 的方式来处理，而是先预检查，比如：NullPointerException处理

反例：

```
try {
  obj.method()
} catch (NullPointerException e) {
...
}
```

正例：

```
if (obj != null){
   ...
}
```

十、注意异常匹配的顺序，优先捕获具体的异常

注意异常的匹配顺序，因为只有第一个匹配到异常的catch块才会被执行。如果你希望看到，是NumberFormatException异常，就抛出NumberFormatException，如果是IllegalArgumentException就抛出IllegalArgumentException。

反例：

```
try {
    doSomething("test exception");
} catch (IllegalArgumentException e) {
    log.error(e);
} catch (NumberFormatException e) {
    log.error(e);
}
```

正例：

```java
try {
    doSomething("test exception");
} catch (NumberFormatException e) {
    log.error(e);
} catch (IllegalArgumentException e) {
    log.error(e);
}
```

理由：

- 因为NumberFormatException是IllegalArgumentException 的子类，反例中，不管是哪个异常，都会匹配到IllegalArgumentException，就不会再往下执行啦，因此不知道是否是NumberFormatException。所以需要优先捕获具体的异常，把NumberFormatException放前面~