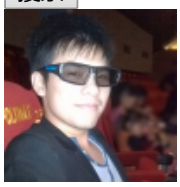


爱捣鼓 关注此人
关注(0) 粉丝(10) 积分(7)
I'll try it out
.发送私信 .请教问题

博客分类

- 编程语言(5)
- WEB技术(1)
- 软件概念(0)
- Linux命令与Shell(2)
- 移动开发(3)
- UI设计(0)
- 其它(0)

阅读排行

1. 1. Java异常处理终结篇——如何进行Java异常处理设计
2. 2. 各语言设计思想的独特之处：C/C++、Java、Python、Objective C、Groovy
3. 3. 轻松理解Java动态代理
4. 4. Andorid Activity的本质是什么
5. 5. 轻松理解正则表达式
6. 6. 用剪纸类比Android View的绘制流程
7. 7. 强大不代表完美——C++几个不方便的地方。
8. 8. 轻松理解AOP(面向切面编程)

最新评论

- @MOYUN_YES：不错 查看»
- @幻の上帝：引用来自“爱捣鼓”的评论 引用来自“幻の上帝”... 查看»
- @爱捣鼓：引用来自“幻の上帝”的评论 引用来自“爱捣鼓”... 查看»
- @幻の上帝：引用来自“ZeroOne”的评论 引用来自“幻の上帝”... 查看»
- @幻の上帝：引用来自“爱捣鼓”的评论 引用来自“幻の上帝”... 查看»
- @爱捣鼓：引用来自“幻の上帝”的评论 搞了半天就没提到e... 查看»
- @爱捣鼓：引用来自“ZeroOne”的评论 看了楼主写的文章， ... 查看»
- @甘薯：看了楼主写的文章， 感觉楼主写的过杂， 其实只要... 查看»
- @甘薯：引用来自“幻の上帝”的评论 搞了半天就没提到e... 查看»
- @幻の上帝：搞了半天就没提到exception safety。 啥能处理就... 查看»

访客统计

- 今日访问：4
- 昨日访问：5
- 本周访问：4
- 本月访问：70
- 所有访问：2868

空间 » 博客 » 编程语言

原 荐 顶 Java异常处理终结篇——如何进行Java异常处理设计

发表于1年前(2014-03-03 09:17) 阅读（1256）| 评论（14） 98人收藏此文章, 我要收藏
赞4

摘要 使用Java异常的人很多，但能合理使用的却不多，Java异常处理设计是一个冷门的话题，但好的异常设计会让程序有质的变化，所以本文从各个方面分析便总结了，在Java程序中，如何进行异常设计。...

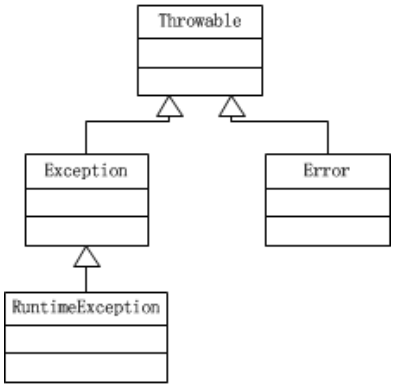
Java 异常处理 设计 异常 原则

有一句这样话：一个衡量Java设计师水平和开发团队纪律性的好方法就是读读他们应用程序里的异常处理代码。

本文主要讨论开发Java程序时，如何设计异常处理的代码，如何时抛异常，捕获到了怎么处理，而不是讲异常处理的机制和原理。

在我自己研究Java异常处理之前，我查过很多资料，翻过很多书籍，试过很多搜索引擎，换过很多英文和中文关键字，但是关于异常处理设计的文章实在太少，在我研究完Java异常处理之后，我面试过很多人，也问过很多老员工，极少碰到对Java异常有研究的人，看来研究这个主题的人很少，本文内容本是个人研究异常时做的笔记，现整理一下与大家一起分享。

首先我们简单的回顾一下基础知识，Java中有两种异常，严格的说是三种，包含四个类，层次图如下：



Throwable是一个可抛类，只有其子类可以被关键字throw抛出，请勿直接继承本类，Error是表示系统级错误，如内存耗尽了，我们一般情况下不用管，Exception是所有异常的父类，所以他的子类，除了RuntimeException及其子类，是属于编译时异常，这种异常必须在代码里被显示的捕获语句包住，否则编译不过，而RuntimeException及其子类表示运行时异常，不强制要求写出显示的捕获代码，但如果没有被捕获到，则线程会被强制中断。

我们主要关注后两种，他们的特点已领教了，下面我们通过回答问题的方式来分析异常设计，在开始之前，请确保你已经知道使这两种异常：

捕获到了编译时异常怎么处理：

这个话题恐怖是最古老的啦，网上的文章多数都是讨论这个话题，但这些文章大部分只是给了几条禁止的原则，他们是：1）不要直接忽略异常；2）不要用try-catch包住过多语句；3）不要用异常处理来处理程序的正常控制流；4）不要随便将异常迎函数栈向上传递，能处理尽量处理。他们都对，但是要做异常处理的设计，信息还是不够，比如第一条他只是告诉了不要忽略，但没有告诉我们怎么处理，所以很多人直接e.printStackTrace()了，这种处理比直接忽略是好一点，但还不够好。对于第二条，他的理由是避免耗资源很大，不过“过多语句”这句话描述的太模糊了，没说明到底多少才算过多，以致于很多人的try-catch语句只包住会抛编译时异常的那一行代码，如果一段代码中有多行代码会抛编译时异常，那这一段代码中可能有多个try-catch语句块，像这样：

```
LLJTran llj = new LLJTran(file);
try {
        llj.read(LLJTran.READ_INFO,true);
```

```
} catch (LLJTranException e) {
        // ...
}

// ...

OutputStream out = null;
try {
        File out = new File(file.getPath()+"_bak.jpg");
        llj.xferInfo(null, out, LLJTran.REPLACE, LLJTran.REPLACE);
} catch (IOException e) {
        // ...
}

// ...

try {
        out.close();
} catch (IOException e) {
        // ...
}
```

这样有什么坏处呢，到处都是异常处理的代码，很容易给人造成困惑，很难找出哪些是正常流程的代码，而且还违背了Java异常机制的初衷，Java异常机制是为了把异常处理的代码与正常流程的代码分开，避免程序中出现过多的像传统程序那样的非法值判断语句，以致于扰乱了正常流程。但上述代段充斥着try-catch语句块，已经扰乱了主流程，并极大影响了可读性。

try-catch既不能包太多代码，又不能包太少，那应该包多少才适合呢，这个问题我查过的资料中都没有提，我的个人建议是包住逻辑关系紧密的代码，比如打开文件，读取文件，关闭文件，我认为就是逻辑关系紧密的代码，如果你发现包住的代码很多，可以封装一些方法，如读取文件的代码很长就应该封装成一个方法，这个方法可以申明IOException，（其实读文件的细节本来属于低层逻辑，打开，读取，关闭才属于同层逻辑，如果读取代码很短，初期为了省事才不封成读取细节的代码，不过后期可以重构并封装成方法，这是《重构·改善继有的代码设计》一书中的思想——软件应该不断的重构和加善）。这样才能达到把异常代码与正常流程代码分离的目的。

第3）条没问题，第4）条也有问题，“不要随便”很模糊，那什么时候才能向上传递呢。

吐槽完了，我们现在来说说到底该如何处理捕获到的编译时异常：

一、恢复并继续执行：这个结果是最完美的，也是编译时异常出生的目的——捕获异常，并恢复继续执行程序。所以如果你捕获了一个异常是先尽力恢复，这种情况其实就是在主方案行不通时，用备选方案，而且主方案能否行通不能事先知道，必须执行的时候才能知道，所以在一般情况下，备选方案比主方案要的运行结果要差。比如一个视频程序，它要调用一个下载节目列表的方法，可能如下：

```
InputStream download() throws IOException {
        // ...
}
```

但服务器不保证总是可用，有可能被攻击了，有可能其它原因，因为是个意外事件，所以又不可能事先知道，于是异常就发生在执行过程中，幸好客户端有备选方案，它在本地保存了一个默认列表，当服务器不可用时，就加载本地列表，所以客户端对这个异常的处理可以如下：

```
public void loadProgramList() {
        InputStream inputStream;
        try {
                inputStream = download();
        } catch (IOException e) {
                // Log this exception
                System.out.println("The server occurred errors");
                // Use the local file
                inputStream = openLocalFile();
        }

        //...
}

private InputStream download() throws IOException {
        // ...
}

private InputStream openLocalFile() {
        // ...
}
```

可惜的是，不是任何时候的异常都可以恢复，反而一般情况是不能恢复的。

二、向上传播异常：向上传播就是在本方法上用throws申明，本方法里的代码不对某异常做任何处理。如果不能用上述恢复措施，就检查能不能向上传播，什么情况下可以向上传播呢？有多种说法，一种说法是当本方法恢复不了时，这个说法显然是错误，因为上层也不一定能恢复。另外还有两种说法是：1.当上层逻辑可以恢复程序时；2.当本方法除了打印之外不能做任何处理，而且不确定上层能否处理。这种两种说法都是正确的，但还不够，因为也有的情况，明确知道上层恢复不了也需要上层处理，所以我认为正确的做法是：当你认为本异常应该由上层处理时，才向上传播。不过这得根据你程序的设计来灵活思考，比如你的类设计了一个上层方法集中处理异常，而下层有一些private方法只是简单的用throws申明。当上层方法捕获到异常时，虽然不能恢复执行，但可以做一些处理，如转换成便于阅读的文本，或者用下面讨论的转译。

三、转译异常：转译即把低层逻辑的异常转化成为高层逻辑的异常，因为有可能低层逻辑的异常在高层逻辑中不能被理解，主要实现是新写一个Exception的子类，然后在低层逻辑捕获异常，改抛这个新写的异常，比如刚刚那个视频程序，他的主流程可能是：1.加载节目列表，2.显示播放节目。而加载节目列表子流程又包含读取节目文件、解析节目文件、显示节目列表。而读取节目文件有可能出现IO异常（有可能本地和网上的文件都读不了了），解析节目文件可能出现解析异常，这时如果把这些异常，直接向上传播，变成这样，你觉得合理吗：

```
public  void  mainFlow()  {
        //  1.load  program  list
        try  {
                loadProgramList();
        }  catch  (IOException  e)  {
                //  I  don't  understand  what  is  this  exception.
        }  catch  (ParseException  e)  {
                //  I  don't  understand  what  is  this  exception.
        }

        //  2.play  program
        //  ...
}

public  void  loadProgramList()  throws  IOException,  ParseException  {
        //  1.Read  program  file
        InputStream  inputStream;
        try  {
                inputStream  =  download();
        }  catch  (IOException  e)  {
                //  Log  this  exception
                System.out.println("The  server  occurred  errors");
                //  Use  the  local  file
                inputStream  =  openLocalFile();        //Maybe  throw  IOException.
        }

        //  2.Parse  program  file
        parserProgramFile(inputStream);                //Maybe  throw  ParseException.

        //  3.Display  program  file
        //...
}
```

由于loadProgramList将两个可能的异常向上传播，在mainFlow里，必须显示捕获这两个异常，但在mainFlow根本就不能理解这两个异常代表什么，mainFlow里只需要知道加载节目列表异常就可以了，所以我们可以写一个异常类LoadProgramException代表加载节目异常，并在loadProgramList里抛出，于是代码变成这样：

```
public  void  mainFlow()  {
        //  1.load  program  list
        try  {
                loadProgramList();
        }  catch  (LoadProgramException  e)  {                //  look  at  here
                //  ...
        }

        //  2.play  program
        //  ...
}

public  void  loadProgramList()  throws  LoadProgramException  {                //  look  at  here
        //  1.Read  program  file
        InputStream  inputStream  =  null;
        try  {
                inputStream  =  download();
        }  catch  (IOException  e)  {
                //  Log  this  exception
                System.out.println("The  server  occurred  errors");
                //  Use  the  local  file
                try  {
                        inputStream  =  openLocalFile();
```

```
                } catch (IOException e1) {
                        throw new LoadProgramException("Read program file error.", e1);              // look at here
                }
        }

        // 2.Parse program file
        try {
                parserProgramFile(inputStream);
        } catch (ParseException e) {
                throw new LoadProgramException("Parse program file error.", e);                      // look at here
        }

        // 3.Display program file
        //...
}

// ...

class LoadProgramException extends Exception {
        public LoadProgramException(String msg, Throwable cause) {
                super(msg, cause);
        }
        // ...
}
```

注意：LoadProgramException构造函数的第一个参数是代表原因，用于组成异常链，异常链是一种机制，异常转译时，保存原来的异常，这样当这个异常再被转译时，还会被保存，于是就成了一条链了，包含了所有的异常，所以你可以看到这样的异常打印：

```
Exception in thread "main" java.lang.NoClassDefFoundError: graphics/shapes/Square
        at Main.main(Main.java:7)
Caused by: java.lang.ClassNotFoundException: graphics.shapes.Square
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        ... 1 more
```

这个异常链中就是包含了两个异常，最前面是顶级异常，后面再打印一个Cause by，然后再打印低一层异常，直到打印完所有的异常。

另外，主流程中还有一个播放流程也可以定义一个播放异常的类，再做这样的转译处理，但是，如果流程多，是不是得写多个异常类呢，有人建议是每个包定义一个异常类，但并不是绝对的，这个细粒度还要根据具体的程序逻辑来决定，这种把握能力就要靠经验了，这可能就是架构师的过人之处了。

四、改抛为运行时异常：这个很好玩，也是一条很方便的处理手法（我常用，我用这个还发现了一个Android系统的bug），即当你捕获到异常时，重新抛出，这跟转译很相似，有一点区别，这里抛的是运行时异常，而转译抛的是编译时异常。那什么时候使用这个手法呢？简单的说就是当某个异常出现时，你必须让程序挂掉。解释一下：如果某个异常情况一旦出现，程序便无法继续执行，而且你明确知道本方法和上层逻辑做不出任何有意义的处理，你只能让程序退出。所以你就抛一个运行时异常让程序挂掉。举个例子，比如在加密通信中，服务器捕获到了一个非法数据异常，这是无法恢复的，而且就是抛一运行异常，让线程挂掉，连接便会自动中断。

五、记录并消耗掉异常：这个手法就是把异常记录下来（到文件或控制台）然后忽略掉异常，有可能随后就让本方法返回null，这个手法一般用在不是很严重的异常，相当于是warning级别的错误，出现这个异常对程序的执行可能影响不太，比如程序的某个偏好设置文件（如窗口位置，最近文件等）损坏，但这个文件信息很少，程序只要使用默认配置即可。

有没必要显示捕获运行时异常：

运行时异常一般是不需要捕获的，因为它的目的就是让程序在无法恢复时挂掉，但是也有特殊需求，比如你要收集所有的未捕获异常记录，可能用于统计，也可能用于将来调试。还有其它原因使你不想让程序直接挂掉，比如你想把友好信息告诉用户。

什么时候需要抛异常：

马上就要讨论如何抛异常了，但在必须先知道，什么时候需要抛异常，简单的说就是遇到一个异常情况，这是一个模

棱两可的问题，就像美不美这个问题一样，我几种说法，你看你能理解哪一种，一种是正常情况的反面，即非正常情况，那什么是非正常情况呢，这也是仁者见仁，智者见智，比如说读到文件尾，这个算正常还是异常呢，都说得过去，所以这里给一个判断方法做为参考，如果是一个典型情况，就不当成是异常，所以读到文件尾就没有被当成一个异常，返回了-1。还有一种说法是，程序执行的必要条件不能成立，使得本方法无法继续履行自己的职责。这两种说法都不错，你都可以用，而且覆盖了大部分情况。

何时选用编译时异常：编译时异常是Java特有的，其它语言没有，刚出来时很流行，所以你可以看到流处理包里充斥着IOException，但经过多年的使用，有人觉得编译时异常是一种实验性错误，应该完全丢弃，说这个话的人就是《Think In Java》的一书的作者Eckel，我认为这种说法太绝对了，关于这个是与否也有很大的争论。《Effective Java》一书的作者则认为应避免不必要的编译时异常，因为你抛编译时异常会给强制要求调用者捕获，这会增加他的负担，我是这一观点的支持者。那到底何时抛编译时异常呢？当你发现一个异常情况时，检查这两个条件，为真时选用编译时异常：一、如果调用者可以恢复此异常情况，二、如果调用者不能恢复，但能做出有意义的事，如转译等。如果你不确定调用者能否做出有意义的事，就别使编译时异常，免得被抱怨。还有一条原则，应尽最大可能使用编译时异常来代替错误码，这条也是编译时异常设计的目的。另外，必须注意使用编译时异常的目的是为了恢复执行，所以设计异常类的时候，应提供尽量多的异常数据，以便于上层恢复，比如一个解析错误，可以在设计的异常类写几个变量来存储异常数据：解析出错的句子的内容，解析出错句子的行号，解析出错的字符在行中的位置。这些信息可能帮助调用恢复程序。

何时选用运行时异常：首先，运行时异常肯定是不可恢复的异常，否则按上段方法处理。这个不可恢复指的是运行时期不可恢复，如果可以修改源代码来避免本异常的发生呢，那说明这是一个编程错误，对于编程错误，一定要抛运行时异常，编程错误一般可以通过修改代码来永久性避免该异常，所以这种情况应该让程序挂掉，相当于爆出一个bug，从而提醒程序员修改代码。这种编程错误可以总结一下，API是调用者与实现者之间的契约，调用者必须遵守契约，比如传入的参数不允许为空，这一点是隐含契约，没必要明确写出来的，如果违反契约，实现者就可以抛运行时异常，让程序挂掉以提醒调用者。

其它情况是否应使用运行时异常，上面提到过，就是谁都无能为力的异常情况，还有就是你不确定到底能不能恢复，除此之外，你可以这样判断：如果你希望程序挂掉，就用运行时异常。需要说明的是，请尽量使用系统自带异常，而不是新写。网上还有一条建议是使用运行时异常时, 一定要将所有可能的异常写进文档。这认为只要把不常用的写上即可，像NullPointException每个方法都有可能抛，但没必要每个方法都写说明。

将编译时异常重构成运行时异常：

你可能手头上有一份以前的代码，大量的使有了编译时异常，但很多都是没有必要的编译时异常，导致调用上不方便，《Effective Java》里有一种方法可以将编译时异常转为运行时异常：将原来抛编译时异常的方法，拆成两个方法，其中一个是用来指示异常是否为发生，即将以下代码：

```
// Invocation with checked exception
try {
obj.action(args);
} catch(TheCheckedException e) {
// Handle exceptional condition
...
}
```

改为这样：

```
// Invocation with state-testing method and unchecked exception
if (obj.actionPermitted(args)) {
obj.action(args);
} else {
// Handle exceptional condition
...
}
```

步骤是：1）将原来方法foo的异常申明删掉，并在实现里面改抛为运行时异常；2）添加一个方法isFoo，返回一个布尔值指示是否会有异常情况出现；3）在foo调用前加一个if语句，判断isFoo的返回值，如果为真才调用foo，否则不调用；4）删掉调用处的try-catch。

UI层处理异常的注意点：

UI层和其下逻辑层的区别是UI层的出错信息是被用户看，而其下层逻层出错信息是被程序员看到，用户可不希望看一个打印的异常栈，更不希望程序无缘无故挂掉，用户希望看到友好的提示信息。为到达这一目的，我们可以设一个屏障，屏障可以捕获所有遗漏的异常，从而阻止程序直接挂掉，屏障当然恢复不了运行，但可以记录错误便于日后调试，还可以输出友好信息给用户。Spring和Struts就有这样的处理。

还有一点需要注意，用户的传入参数出现非法的概率很高，所以控制层接受到参数时一定要校验，而不是原封不动的传到其低层模块。

经历了一周的熬夜，总算把异常处理总结归纳成文了，但由于文章太长，肯定有一些错误和语言不精炼的地方，我会仔细检察并及时改正，希望本文对大家有一定的帮助。

附录

在我查过的资料中，以《Effective Java》书中对异常处理设计的研究得最系统，本文很多思想来自于它，下面我把其中的几条原则翻译（非直译）并贴上：

第57条：只对异常情况使用异常。（说明：即不要用异常处理控制正常程序流）。

第58条：对可恢复异常使用编译时异常，对编程错误使用运行时异常。

第59条：应避免不必要的编译时异常：如果调用者即使合理的使用API也不能避免异常的发生，并且调用者可以对捕获的异常做出有意义的处理，才使用编译时异常。

第60条：应偏好使用自带异常

第61条：抛出的异常应适合本层抽象（就是上面说的转译）

第62条：把方法可能抛的所有异常写入文档，包括运行时异常

第63条：用异常类记录的信息要包含失败时的数据

第64条：力求失败是原子化的（解释：就是如果调用一个方法发生了异常，就应该使对象返回调用前的状态）

第65条：不要忽略异常

参考资料：

Effective Java, 2nd.Edition

Effective Java Exceptions, 译文可参考这里或这里

- « 上一篇
- 下一篇 »

评论14

- 
1楼：小黑001 发表于 2014-03-03 17:19 回复此评论
有demo提供更好了

- 2楼：爱捣鼓 发表于 2014-03-03 18:51 回复此评论

引用来自“小黑001”的评论

有demo提供更好了

你说的有道理，如果代码太多，但我担心有童鞋看了会想睡觉，要想个办法把代码与文章分离，然后又很方便打开代码。

- 
3楼：flyed Android 发表于 2014-03-03 23:45 回复此评论
异常一般是为了修复可能出现的问题，但在大多数情况下这种问题都是不可修复的，所以在顶级处理层都会抛出RuntimeException

- 4楼：爱捣鼓 发表于 2014-03-04 00:05 回复此评论

引用来自“flyed.liu”的评论

异常一般是为了修复可能出现的问题，但在大多数情况下这种问题都是不可修复的，所以在顶级处理层都会抛出RuntimeException

我认同前半部分，最后一句，我觉得不一定，有的顶级处理层对不严重的异常会忽略，如软件的偏好配置文件损坏，不会影响执行，但也不能恢复。

- 
5楼：爱捣鼓 iPhone 发表于 2014-03-04 00:16 回复此评论
还有的顶级处理层会设置屏障，如spring和struts，所以web服务器遇到异常也不会挂，只是报错。

- 
6楼：幻の上帝 发表于 2014-03-04 08:17 回复此评论
搞了半天就没提到exception safety。
啥能处理就尽量处理，大部分撸Java的果然连exception neutrality的概念都没有。

- 7楼：甘薯 发表于 2014-03-04 11:11 回复此评论

引用来自“幻の上帝”的评论

搞了半天就没提到exception safety。
啥能处理就尽量处理，大部分撸Java的果然连exception neutrality的概念都没有。

去百度了一下exception safety，因为搞java大多数情况下不需要担心资源泄露的问题（除了处理编译时异常和外部资源连接），所以你提到的这个东西基本上只适合于编译型的语言。

- 8楼：甘薯 发表于 2014-03-04 11:13 回复此评论

看了楼主写的文章，
感觉楼主写的过杂，
其实只要指出，
1.异常是什么分哪几种都有什么作用
2.系统异常使用场合，自定义异常的使用场合
3.异常需要携带的信息和捕获的地点。
我觉得就够了。:)

- 9楼：爱捣鼓 发表于 2014-03-04 11:42 回复此评论

引用来自“ZeroOne”的评论

看了楼主写的文章，
感觉楼主写的过杂，
其实只要指出，
1.异常是什么分哪几种都有什么作用
2.系统异常使用场合，自定义异常的使用场合
3.异常需要携带的信息和捕获的地点。
我觉得就够了。:)

非常谢谢你的建议，不过你说的这种思路网上已有很多优秀的文章，我再写一篇可能还没有人家好，本文的思路是：捕了怎么办，自己何时需要抛，抛哪种。还有oschina的格化式不好，看起有点乱，你可以看看我CSDN上的同篇文章，那里有着色和加粗，可能格化好点，内容一样，地址：http://blog.csdn.net/yanquan345/article/details/19633623

- 10楼：爱捣鼓 发表于 2014-03-04 11:46 回复此评论

引用来自“幻の上帝”的评论

搞了半天就没提到exception safety。
啥能处理就尽量处理，大部分撸Java的果然连exception neutrality的概念都没有。

小伙子，不要冲动，你这句话好像一棒子打到了大部分Java程序员，你说的那个在C++里有这个要求，在Java里需求不太，《Effective Java》书上也没有提这两个概念。

- 1
- 2
- >

插入： 表情 开源软件

发表评论

关闭插入表情

关闭相关文章阅读

- 2014/11/27 java中的异常及异常处理

- 2013/08/30 Java Web整体异常处理
- 2012/11/05 Java基础—异常处理总结
- 2013/11/21 java异常处理好习惯