高质量的软件产品离不开“异常处理编程机制”

作者：王胜祥
时间：2005/06

相关专栏：http://51cmm.csai.cn/ExpertColumn/No013.asp

软件产品的质量

- 软件产品的规模越来越大，软件产品的质量也因此变得愈难以控制。

- 软件产品的质量决定了软件产品的生命。保证和提高软件产品的质量依赖于许多种措施！例如：严谨而规范化的项目管理，使用成熟的技术，软件产品分析、设计和编程人员的综合素质，以及阶段化评审和软件测试等。

- 软件测试对提高软件产品的质量起着至关重要的作用。但是，任何一种（或多种）软件测试方法并不能检测出软件产品中所有的bug，一种好的软件测试方法只能是说“用最少的资源来发现出更多的软件缺陷”。

- 因此，高质量的软件产品，它必须具备一定的“容错性”。

目前许多产品的质量令人堪忧

- 进程内部死锁了，不工作了！但外部毫无觉察。

- 运行过程中出现崩溃，这种问题简直让客户完全无法接受，例如，编辑word文档时，突然出现了一个“致命错误”，导致部分为保存的数据丢失。又例如，需要7x24小时不间断工作的服务程序（守候进程）更不能出现意外崩溃。

- 纵多的开源软件，由于缺乏系统的测试，以及其它多方面的原因，导致它们的产品质量更不容乐观。例如，Linux下的资源管理器，web浏览器，等其它众多程序，意外崩溃的现象屡见不鲜。

软件产品“容错性”的提高离不开“异常处理编程机制”

- 人非圣贤，孰能无过。的确，软件产品中存在一些错误，在所难免！

- “谁人无过，过而能改，善莫大焉”这句话应用到软件产品的开发中来，不完全对！因为，客户（或实际的应用环境）不能接受一个存在严重缺陷的软件产品（尤其是不定期的崩溃现象）

- “异常处理机制”可以允许我们犯一些错误，也即使软件产品具备一定的“容错性”

异常处理编程机制简介

- “异常处理的编程和面向对象的方法”，是软件程序设计发展史上其中最重要的两项革新技术。现代程序设计语言拥有的一个重要的特性就是能较好地支持异常的处理（Exception Handling）。她就像一位美丽而优雅的公主，帮助程序员写出来的代码总是那样的整齐美观、层次清晰；同时它好像还是一位贤惠能干的贤内助，总能帮你料理好由于考虑不全所留下的多多少少的意外事件，她在背后默默的支持你的一切，使你写出来的作品是那样的高效、安全和完美。

- “异常处理编程”对程序设计领域中所起的作用和意义，等同于计算机硬件系统中的“中断”设计，中断是计算机硬件执行过程中的出现的异常情况。等同于操作系统中的“信号”设计，同样它是操作系统在运行过程中的例外情况。所以说，异常处理对于对程序设计领域中的意义非同一般。

“异常处理”与“产品质量”

- 异常处理编程机制，为提高软件产品质量有积极的意义。它主要体现在：
  - ✓ 代码易于组织了，也易于它人理解和阅读了，毫无疑问，程序员在编写代码时由于不小心所导致的bug也当然就更少了；
  - ✓ 有了异常处理机制，对于那些复杂的、庞大的业务逻辑也更容易组织；而且异常的声明可以成为函数接口的一部分，这能便于错误的统一处理；
  - ✓ 经济和社会学领域中的“2/8法则”，同样适合与软件开发领域，也即程序中80%的代码都是去完成了可能与业务无关的工作（错误处理），异常处理编程机制提供了一种最为严谨、也最为灵活的错误处理和错误恢复的方法；

“异常处理”与“产品质量”

- 异常处理编程机制，为提高软件产品质量有积极的意义。它主要体现在：（续）

  ✓ 异常处理编程机制，是提高软件产品容错性最为有效的方法；

  ✓ “测试”为了发现软件产品中的bug，而“调试”是去定位测试中发现的bug，并解决修复这个bug。异常处理机制能够为调试提供更多log信息，便于bug的定位和定性；

  ✓ 软件产品的“容错性”固然重要，但是，这并不以为它将成为软件产品中bug的避风港。异常处理机制，可以用最为合理，也最为严谨的方式分类纪录下各种错误log信息，这位后期的软件产品的有效维护提供了事实依据。

正确使用C++异常处理（1）

- 用对象来描述程序中出现的异常

✓ 面向对象的实现中，一般都很好地实现了对象的RTTI技术，如果异常用对象来表示，那么就可以很好完成异常对象的数据类型匹配，还有就是函数的多态。利用面向对象的继承性特点，可以便于异常错误的集中处理（利用catch一个基类）；

✓ 面向对象的实现中，一般都很好的实现了对象的构造、对象的销毁、对象的转存复制等等，所以这也为异常处理模型中，异常对象的转存复制和对象销毁提供了很好的支持，容易控制。

正确使用C++异常处理（2）

- 使用“层次式”的多个try-catch嵌套结构，或并列的多个catch结构来进行异常的分类处理

- 在每个线程的顶层（toplevel）函数中，使用catch(...)语法来捕获所有异常

正确使用C++异常处理（3）

- 强烈建议异常对象按引用方式被传递，为什么呢？如下：

	按值传递	引用传递	指针传递
语法	catch(std::exception e)	catch(std::exception& e)	catch(std::exception* e)
如何抛出异常？	①throw exception() ②exception ex; throw ex; ？ throw ex_global;	①throw exception() ②exception ex; throw ex; ？ throw ex_global;	①throw new exception();
异常对象的构造次数	三次	二次	一次
效率	低	中	高
异常对象什么时候被销毁	①局部变量离开作用域时销毁 ②临时变量在 catch block 执行完毕后销毁 ？ catch 后面的那个类似参数的异常对象也是在 catch block 执行完毕后销毁	①局部变量离开作用域时销毁 ②临时变量在 catch block 执行完毕后销毁	异常对象动态地在堆上被创建，同时它也要动态的被销毁，销毁的时机是在 catch block 块中处理完毕后进行
发生对象切片	可能会	不会	不会
安全性	较低，可能会发生对象切片	很好	低，依赖于程序员的能力，可能会发生内存泄漏，或导致程序崩溃
综合性能	差	好	一般
易使用性	好	好	一般

正确使用C++异常处理（4）

- 不要在析构函数中抛出异常，因为：

  ✓ 假如析构函数中抛出了异常，那么你的系统将变得非常危险，也许很长时间什么错误也不会发生；但也许你的系统有时就会莫名奇妙地崩溃而退出了，而且什么迹象也没有，崩得你满地找牙也很难发现问题究竟出现在什么地方；

  ✓ 当在某一个析构函数中会有一些可能（哪怕是一点点可能）发生异常时，那么就必须要把这种可能发生的异常完全封装在析构函数内部，决不能让它抛出函数之外（这招简直是绝杀！呵呵！；

详细资料请参阅：第8集 析构函数中抛出的异常

把系统异常统一到C++的异常处理框架之中（Windows系统）

- 为什么要把系统异常转化为C++类型的异常？

  ✓ 异常主要分为两类，一种是程序在运行中，检测到了一个异常，通过throw关键字显式地抛出的异常；另一类就是系统异常，如存储保护、被0除等，这类异常出现时，将引发一个“中断”事件 ，于是操作系统内核将接管控制权；

  ✓ catch(...)的语法虽然可以捕获系统异常，但是，“异常发生时的许多相关信息”它却什么也没有提供给程序员（包括何种类型的系统异常，出现的地点，以及其它有关异常的信息等等）；

  ✓ C++异常处理是和面向对象是紧密联系的（它二位可是“哥俩好”），因此，如果把系统异常统一到面向对象方法设计的“异常分类”中去，那对程序员而言，岂不是妙哉！美哉！该解决方案真所谓是，即提高了可靠性；又不失优雅！

- 如何实现把SEH类型的系统异常转化为C++类型的异常？

  ✓ 通过“_set_se_translator”函数来设置异常出现时的回调函数，再在回调函数中抛出对应的C++异常

详细资料请参阅：

把系统异常统一到C++的异常处理框架之中（类Unix系统）

- 在类Unix系统中，更需要把系统异常转化为C++类型的异常？
  - ✓ 类Unix操作系统中，对程序运行中出现的“系统错误”的处理，被统一到了信号（Signal）的处理框架之中，而且GCC所实现的C++异常处理框架中，它的catch(...)语法，并不能捕获系统异常。因此，这给C++中异常处理的良好运用打了大大的折扣；
  - ✓ 信号（Signal）处理的编程相对比较麻烦，而且它对系统异常错误的处理也是过于简单和粗暴了一些（往往是直接terminate进程）。这与基于源码级、既灵活也功能强大的C++异常处理模型是无法媲美的！
- 在Unix系统中，如何实现把系统错误转化为C++类型的异常？
  - ✓ 与Windows系统相类似，首先通过“signal”函数来设置相应异常的信号处理的回调函数，再在回调函数中抛出对应的C++异常。

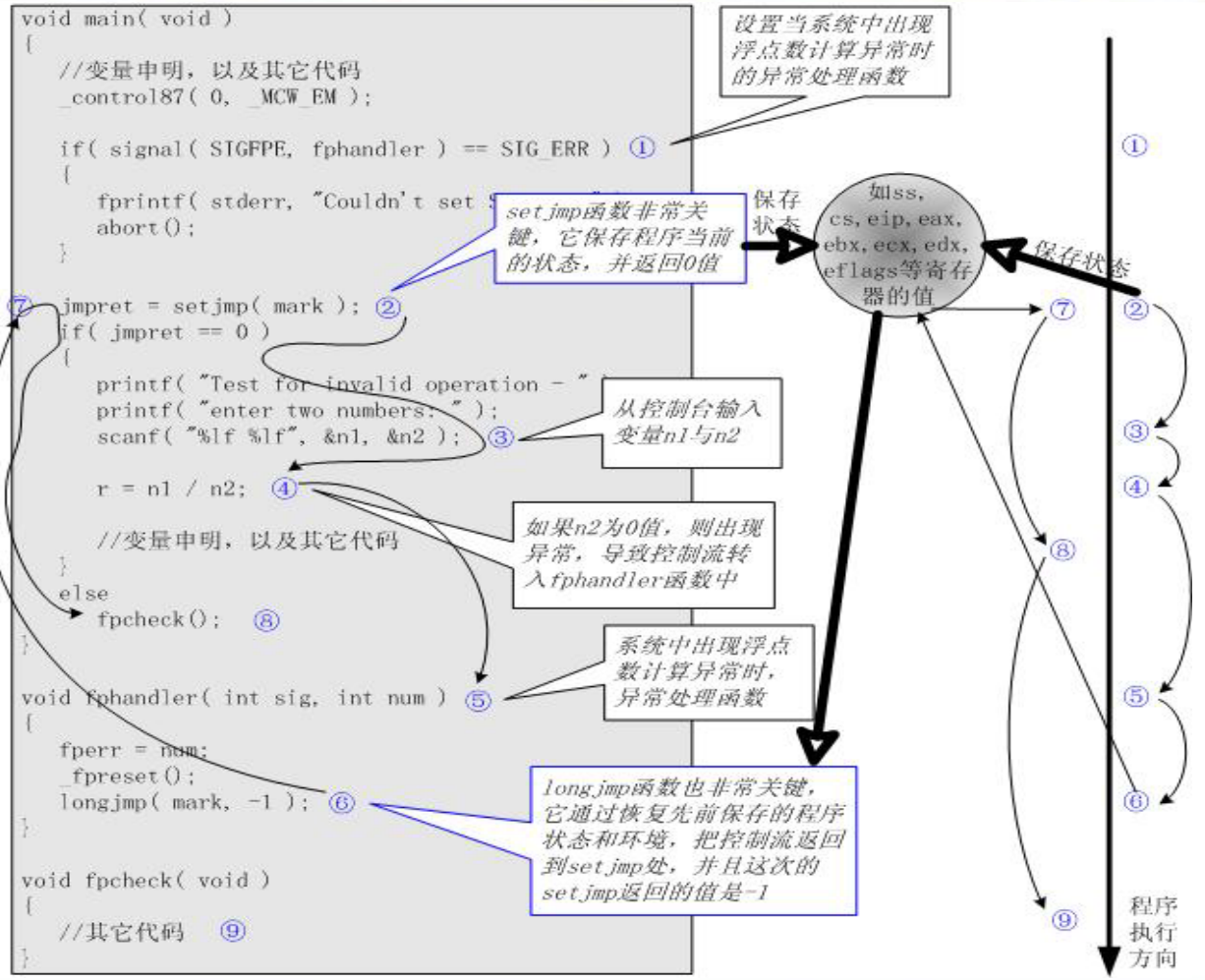
用C写程序也不要忘记使用异常处理

- goto能实现本地跳转，而setjmp()与longjmp()的组合运用，能有效地实现程序控制流的非本地（non-local goto,也即跨函数作用域）跳转；

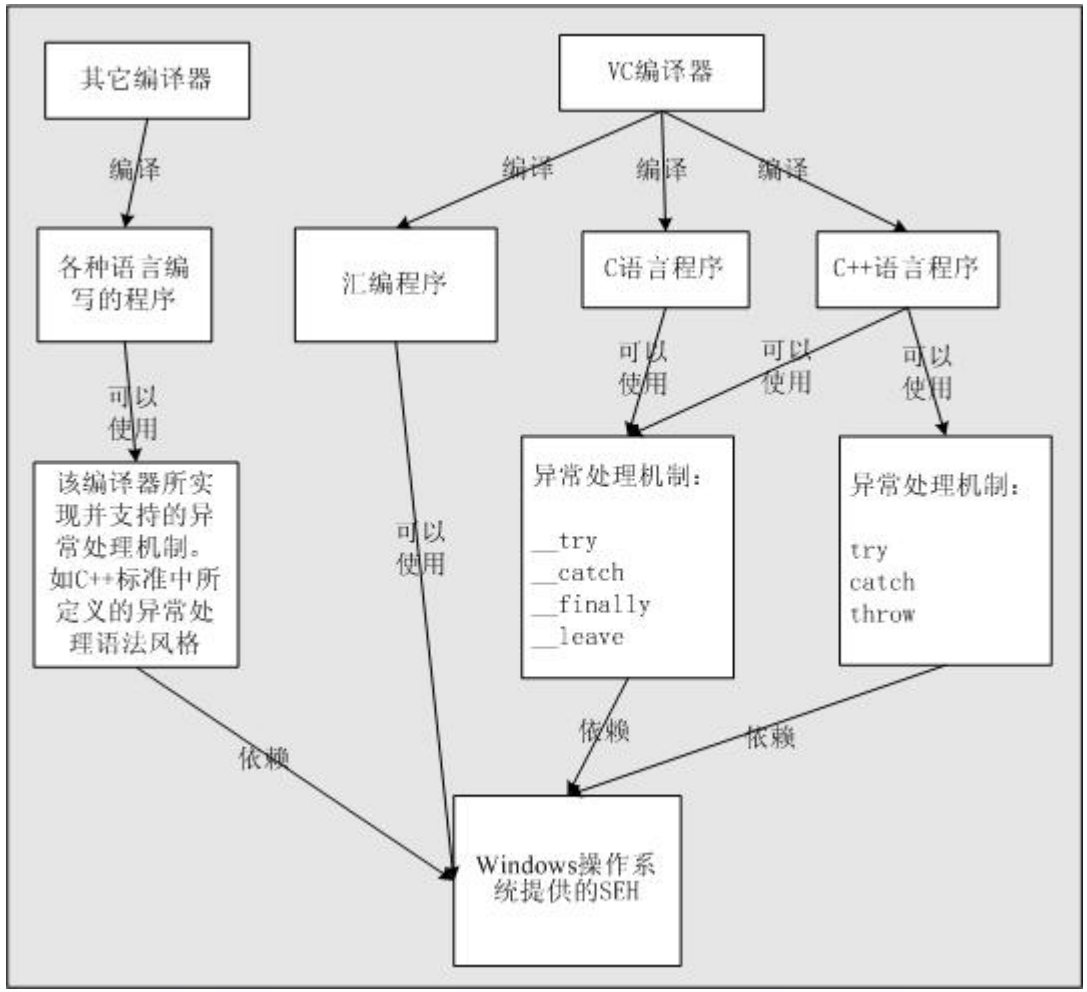
- 与goto语句不同，setjmp()与longjmp()的组合运用，提供了真正意义上的异常处理机制。例如，它能有效定义受监控保护的模块区域（类似于C++中try关键字所定义的区域）；同时它也能有效地定义异常处理模块（类似于C++中catch关键字所定义的区域）；还有，它能在程序执行过程中，通过longjmp函数的调用，方便地抛出异常（类似于C++中throw关键字）。

看一个示例分析

用C写程序也不要忘记使用异常处理



不要忘记强大的SEH


其它编译器
VC编译器
编译
编译
编译
编译
各种语言编写的程序
汇编程序
C语言程序
C++语言程序
可以使用
可以使用
可以使用
可以使用
该编译器所实现并支持的异常处理机制。如C++标准中所定义的异常处理语法风格
异常处理机制：
__try
__catch
__finally
__leave
异常处理机制：
try
catch
throw
依赖
依赖
依赖
Windows操作系统提供的SEH

SEH的强大功能之一

- SEH异常模型中用try-except来实现类似于C++异常模型中的try-catch的功能；

- 与C++异常模型相似，try-except模型中，异常发生后，搜索异常处理模块的规则也是逐级向上进行的。但是稍有区别的是，C++异常模型是按照异常对象的类型来进行匹配查找的；而try-except模型则不同，它通过一个表达式的值来进行判断。如果表达式的值为1（EXCEPTION_EXECUTE_HANDLER），表示找到了异常处理模块；如果值为0（EXCEPTION_CONTINUE_SEARCH），表示继续向上一层的try-except域中继续搜索；如果值为-1（EXCEPTION_CONTINUE_EXECUTION），表示忽略这个异常，注意这个值一般很少用，因为它很容易导致程序难以预测的结果，例如，死循环，甚至导致程序的崩溃等；

- __except关键字后面跟的表达式，它可以是各种类型的表达式，例如，它可以是一个函数调用，或是一个条件表达式，或是一个逗号表达式，或干脆就是一个整型常量等等。最常用的是一个函数表达式，并且通过利用GetExceptionCode()或GetExceptionInformation ()函数来获取当前的异常错误信息，便于程序员有效控制异常错误的分类处理。

- SEH异常处理模型中，异常被划分为两大类：系统异常和软件异常。其中软件异常通过RaiseException()函数抛出。RaiseException()函数的作用类似于C++异常模型中的throw语句。

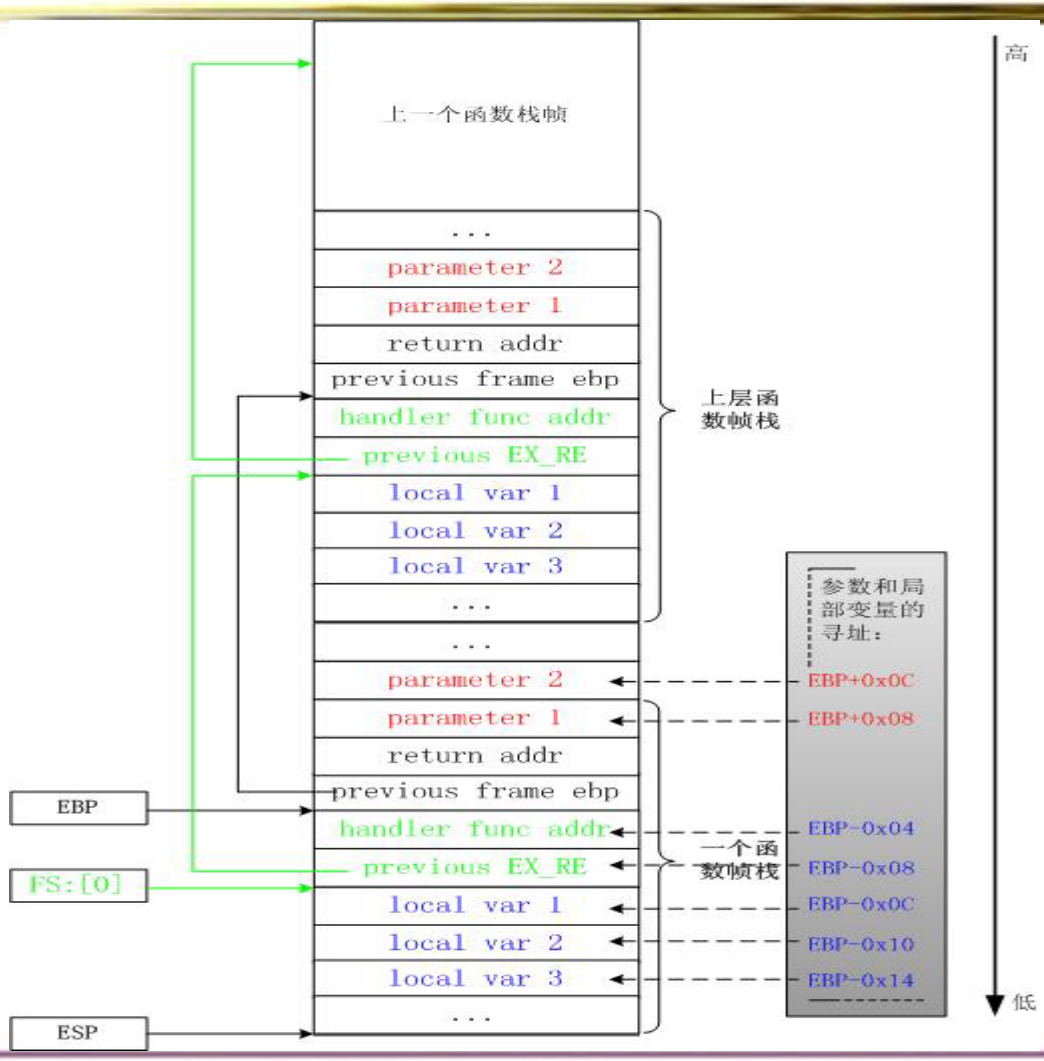
SEH的强大功能之二

- 也许觉得try-except的功能不过如此，那么try-finally的表现一定令众人刮目相看。它才是SEH的精华所在；

- try-finally语句是Microsoft对C和C++语言的扩展，它能使32位的目标程序在异常出现时，有效保证一些资源能够被及时清除，这些资源的清除任务可以包括例如内存的释放，文件的关闭，文件句柄的释放等等。try-finally语句特别适合这样的情况下使用，例如一个例程（函数）中，有几个地方需要检测一个错误，并且在错误出现时，函数可能提前返回；

- goto语句和return语句（在其它少数情况下，break语句以及continue语句）等，它们都可能会导致程序的控制流以非正常的顺序离开__try作用域，此时则会发生SEH的“局部展开”。记住，“局部展开”会带来较大的开销，因此，程序员应该尽可能采用__leave关键字来减少一些不必要的额外开销。

初探SEH的实现


上一个函数栈帧
...
parameter 2
parameter 1
return addr
previous frame ebp
handler func addr
previous EX_RE
local var 1
local var 2
local var 3
...
...
parameter 2
parameter 1
return addr
previous frame ebp
handler func addr
previous EX_RE
local var 1
local var 2
local var 3
...
上层函数帧栈
一个函数帧栈
EBP
FS:[0]
ESP
参数和局部变量的寻址：
EBP+0x0C
EBP+0x08
EBP-0x04
EBP-0x08
EBP-0x0C
EBP-0x10
EBP-0x14
高
低

最为严谨的JAVA异常处理模型

- Java异常处理模型与C++中异常处理模型的最大不同之处，就是在Java异常处理模型中引入了try-finally语法，它主要用于清理非内存性质的一些资源（垃圾回收机制无法处理的资源），例如，数据库连接、Socket关闭、文件流的关闭等；

- 所有的异常都必须从Throwable继承而来，不像C++中那样，可以抛出任何类型的异常。因此，在Java的异常编程处理中，没有C++中的catch(...)语法，而它的catch(Throwable e)完全可以替代C++中的catch(...)的功能；

- 在Java的异常处理模型中，要求所有被抛出的异常都必须要有对应的“异常处理模块”。也即是说，如果你在程序中throw出一个异常，那么在你的程序中（函数中）就必须要catch这个异常（处理这个异常）。但是，对于RuntimeException和Error这两种类型的异常（以及它们的子类异常），却是例外的。其中，Error表示Java系统中出现了一个非常严重的异常错误；而RuntimeException虽然是Exception的子类，但是它却代表了运行时异常；

- 如果一个函数中，它运行时可能会向上层调用者函数抛出一个异常，那么，它就必须在该函数的声明中显式的注明（采用throws关键字，语法与C++类似）。

Java异常处理模型之细节分析

- finally区域内的代码总在return之前被执行；

- 强烈建议不要在finally内部使用return语句。它不仅会影响函数的正确返回值，而且它可能还会导致一些异常处理过程的意外终止，最终导致某些异常的丢失。

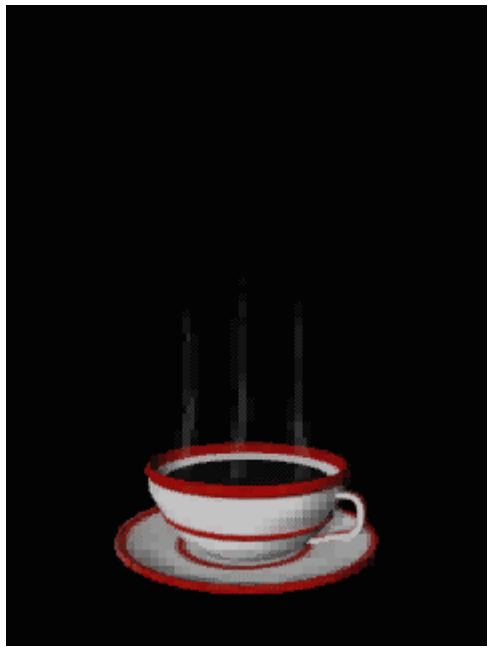
看一个示例分析

Java异常处理模型之细节分析

```
import java.io.*;

public class Trans
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("test的返回值为：" + test());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public static int test() throws RuntimeException
    {
        int ret = 0;
        try
        {
            System.out.println("in try block");

            //这里会导致出现一个运行态异常
            int i=4,j=0;
            ret = i/j;
        }
        catch(RuntimeException e)
        {
            System.out.println("in catch block");
            e.printStackTrace();

            //异常被重新抛出，上层函数可以进一步处理此异常
            throw e;
        }
        finally
        {
            System.out.println("in finally block!");

            //注意，这里有一个return语句
            return ret;
        }
    }
}
```

显示运行结果

```
in try block
in catch block
java.lang.ArithmeticException: / by zero
        at Trans.test(Trans.java:27)
        at Trans.main(Trans.java:10)
in finally block!
test的返回值为：0
```



Thank you for your attention!