基于C++异常处理机制的研究

王方良,汤文成
(东南大学,江苏 南京 210096)

摘 要:异常处理是程序开发的一个重要内容,异常处理的好坏关系到程序的友好程度和系统的稳定性。C++是一种纯面向对象的编程语言,其异常处理机制和普通的编程语言有很多不同的地方,有力地增强了C++程序的健壮性和容错性。文中从介绍其运行步骤入手,通过一个简单的例子,详细阐述了C++异常处理机制的规则和栈展开技术,并扼要地对比了函数调用和异常处理的异同点,以便更好地在C++面向对象程序设计中正确使用异常处理机制。

关键词:C++;异常处理;栈展开
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)01-0128-02

Discussion on the Exception Handling of C++

WANG Fang-liang, TANG Wen-cheng
(Southeast University, Nanjing 210096, China)

Abstract: Exception handling plays a significant role in programming development. Its performance will influence the friendliness and stability of the system. As a complete object oriented programming language, the exception handling mechanism of C++ is very different from other programming language, and effectively improves robustness and tolerance of the programs. This paper began with the introduction of the running steps, and then illustrated the rule of exception handling mechanism and the technology of stack unwinding by a test example. Finally it concisely compared the similarities and differences of function transferring and exception handling in order to make precise use of the mechanism in the object oriented programming.

Key words: C++; exception handling; stack unwinding

面向对象的C++程序设计语言可以编写大型和十分复杂的程序,这样的程序往往会产生一些很难查找甚至是无法避免的运行时错误。当发生运行时错误时,不能简单地结束程序运行,而是退回到任务的起点,指出错误,并由用户决定下一步工作。面向对象的异常处理(Exception Handling)机制便是C++语言用以解决这个问题的有力工具,并成为提高程序健壮性和容错性的重要手段之一。

1 异常处理的运行步骤及规则

1.1 运行步骤

异常表示程序在执行过程中发生的问题,异常的出现很难避免。20世纪80年代末的一项研究表明,一个程序里处理错误的代码可能达到整个代码量的2/3[1],由此可见异常处理在编程中的重要性。基于此C++引入了异常处理机制,它主要包括异常定义、异常抛出和异常捕获及处理三个关键步骤[2]。对应的程序形式分别是try代码块、throw代码块和catch代码块:

```
…;
try{…throw E;}
catch(…){…}
catch(…){…}
…;
```

首先C++用try代码块激活异常处理程序即进行异常定义,try代码块由关键字try后紧跟{}组成,{}中封装了可能导致异常的语句;然后用throw语句抛出可以是任何类型的异常数据对象(当然,throw语句并不一定放在try代码块中,也可在其之前或之后);最后用catch代码块对异常进行捕获和处理。

在这三个步骤中catch代码块较为复杂。catch代码块由三部分组成:关键字catch、圆括号中的异常声明(Exception Declaration)以及复合语句中的一组语句。注意这不是函数,所以圆括号中不是形参,而是一个异常类型声明,可以是类型也可以是对象。catch子句与函数的不同之处在于:它只有一个子句,没有定义和调用之分[3]。使用时由系统按规则自动在catch子句列表中匹配。至少从逻辑上讲,没有函数的定义与调用。关于异常处理与函数的区别,文中后面还有补充,在此不再赘述。

1.2 异常处理实例

下面通过一个具体实例来说明C++中异常处理的

收稿日期:2006-04-06

作者简介:王方良(1977-),男,河南人,硕士研究生,研究方向为程序设计与数据库技术;汤文成,教授,研究方向为CAD/CAE/CAM/PDM/ERP。

实现过程:

```
#include<iostream.h>
int CalFun(int num1,int num2)
{
if(num2==0)
  throw num2; //抛出一个整型异常
else return num1/num2;
}
void main()
{
int a,b;
try{ //可能产生错误的语句块
  cout<<"请输入两个整数:";
  cin>>a>>b;
  cout<<"a="<<a<<'\t'<<"b="<<b<<endl;
  cout<<"a+b="<<a+b<<endl;
  cout<<"a/b="<<CalFun(a,b)<<endl;
  cout<<"a-b="<<endl;
  }
  catch(int) //捕获整型类型的异常
  {
    cerr<<"捕获了除数为 0 的异常!"<<endl;
    cerr<<"程序终止!"<<endl;
  }
}
```

输入 3 和 0 后运行程序,其输出结果为:

$a = 3\ b = 0$

$a + b = 3$

捕获了除数为 0 的异常!

程序终止!

此例中 throw 代码块就没有放在 try 代码块中。当主程序调用函数 CalFun 时产生并抛出一个整型异常,然后由 catch 代码块捕获并进行处理。

从例子中也可看出,一旦异常能被处理,程序就不能在异常被抛出的地方继续,而是执行 catch 子句后的语句,或退出。如本例中异常被处理后,语句"cout<<"a-b="<<endl;"便未被执行。

1.3　异常处理规则

C+ +异常处理机制有一些特殊的规则,在编写程序时应注意:

(1)与普通的语句块一样,try 块内定义的量在块外是不能引用的,只能在 try 块局部域中有效。

(2)编写程序时有一条惯例:把正常的程序与异常处理两部分分隔开来,这样使代码更易于跟踪和维护。其中最清楚的方法是定义函数 try 块,这种方法是把整个函数包括在 try 块中,但 VC+ +6.0 不支持[4]。

(3)catch 子句必须在 try 块之后;而且 try 块后必须紧跟一个或多个 catch 子句,目的是对发生的异常进行处理。

(4)catch 子句括号中的异常声明只能有一个类型;如果在事前不能确定异常声明的类型,则可以用 catch_all 子句来捕获任何异常,其格式为:

Catch(…){//代码块}

其中的 3 个点称为省略号(Ellipsis),{}中的复合语句用来执行指定操作,当然可以包括资源的释放。且 catch_all 子句必须是紧跟在 try 块后的最后一个 catch 处理程序,否则将发生语法错误[2]。

(5)catch 子句可以包含返回语句,也可以不包含返回语句。包含返回语句,则整个程序结束;若不包含返回语句,则执行 catch 列表之后的下一条语句。

2　异常处理核心技术——栈展开

异常处理的关键在于在最适当的地方捕捉和处理异常,使软件能继续正确运行。而 C+ +异常处理机制寻找匹配的 catch 子句有固定的过程:如果 throw 表达式位于 try 块中,则检查与 try 块相关联的 catch 子句列表,看是否有一个子句能够处理该异常,如果有匹配的,则该异常被处理;如果找不到匹配的 catch 子句,则在主调函数中继续查找。如果一个函数调用在退出时带有一个被抛出的异常未能处理,而且这个调用位于一个 try 块中,则检查与该 try 块相关联的 catch 子句列表,看是否有一个子句匹配,如果有,则处理该异常;如果没有,则查找过程在该函数的主调函数中继续进行。即这个过程逆着嵌套的函数调用链向上继续,直到找到处理该异常的 catch 子句。只要遇到第一个匹配的 catch 子句,就会进入该 catch 子句,进行处理,查找过程结束。

在这一过程中,因发生异常而逐步退出复合语句和函数定义,被称为栈展开(Satck Unwinding)。栈展开是异常处理的核心技术。异常对程序的影响通常不仅是在发生异常的那个局部范围中,而且可能逆调用链而上,甚至影响整个任务。因此,异常处理应该在其对程序影响的终结处进行,甚至是在调用该任务的菜单处进行。

退出调用链时必须释放所有资源。随着栈展开,在退出的复合语句和函数定义中声明的局部变量的生命期也结束了。在栈中分配的局部变量占有的资源也被释放,由系统回收。但是如果函数动态获得过资源(包括 new 运算符取得的资源和打开的文件),因异常,这些资源的释放语句可能被忽略,则这些资源将永远不会被自动释放。

在栈展开期间,当一个复合语句(或语句块)或函数退出时,在退出的域中由于某个局部量是类对象,栈展开过程将自动调用该对象的析构函数,完成资源的释放。所以 C+ +异常处理过程本质上反映的是"资源获取是由构造函数实现,而资源释放是由析构函数完成"这样一种程序设计技术[4]。采用面向对象的程序设计,取得资源的动作封装在类的构造函数中,释放资源的动作封装在类的析构函数中,当一个函数带有未处理的异常退出时,函数中这类对象将被自动销毁,资源(包括动态空间分配的资源和打开的文件)得到释放。栈展开过程决不会跳过封装在类的析构函数中的资源释放动作。

(下转第 132 页)

首先进行粗分类,利用主要结构特征判别数字属于哪个特征类;第二阶段,在所属特征类中检查是否能再次分组,如不能再分组则进行细分类,输入 BP 网络进行识别,系统图如图 4、图 5 所示。因为采用了对数字粗分类的方法,所以可以减少查找特征次数,降低网络的规模,提高识别速度。

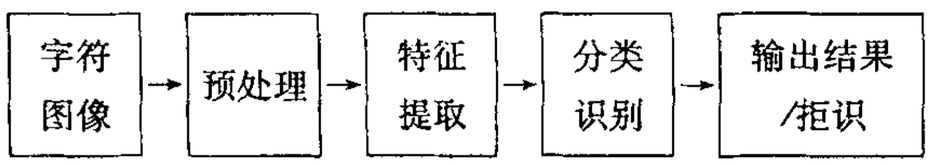

图 4 字符识别流程图

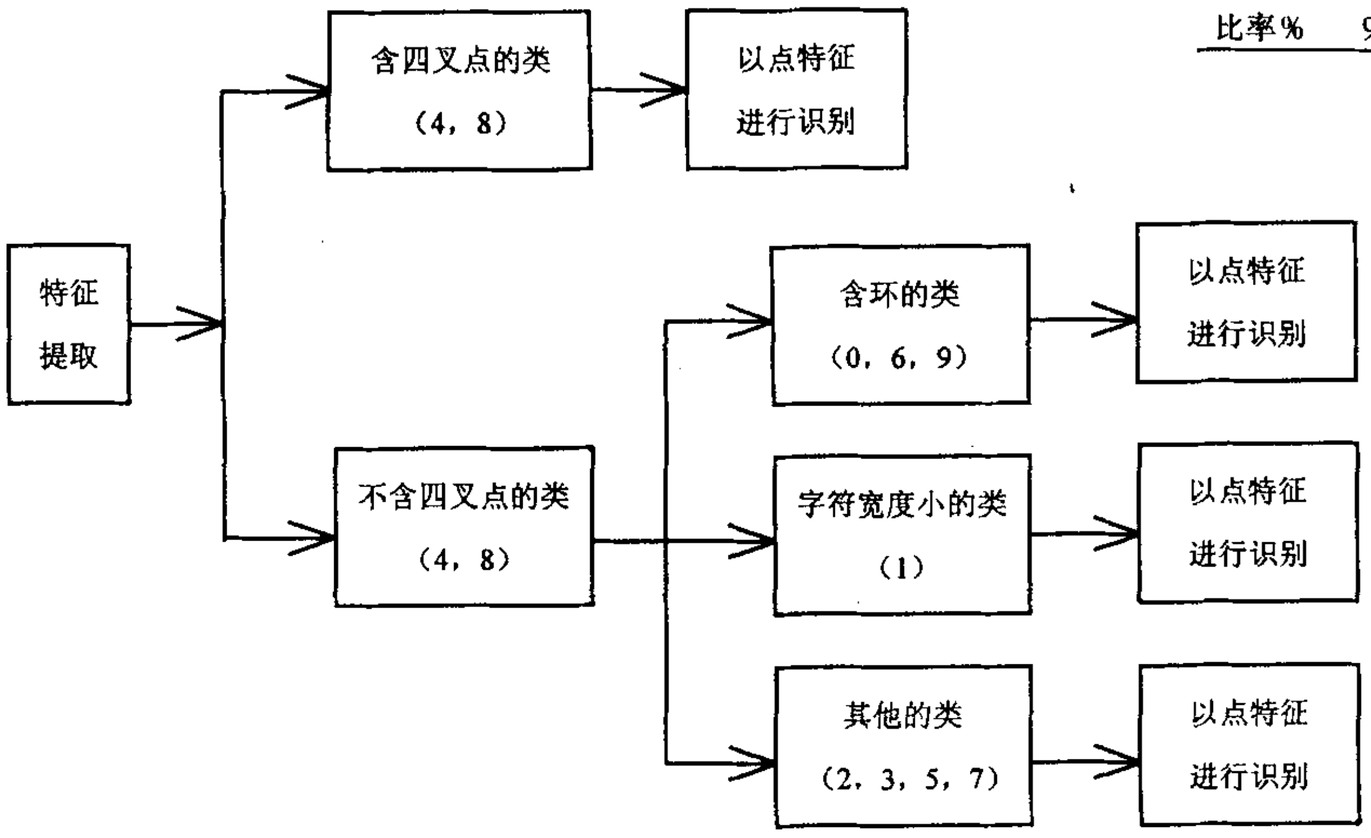

图 5 分类识别流程图

3 结 论

选取了 500 个人的 0～9 的手写体数字,运用以上算法进行 BP 神经网络识别,用 3000 个手写体数字作为训练样本,2000 个其他的样本进行测试,网络收敛后,识别率达到 96% 以上(见表 1)。

表 1 试验结果对比表

类别	正确识别	误识	拒识	识别率%	可靠性%
0	492	3	5	98.40	99.39
1	497	0	3	99.40	100
2	485	7	8	97.00	98.38
3	478	3	19	95.60	99.38
4	480	4	16	96.00	99.17
5	489	2	9	97.80	99.59
6	482	3	15	96.40	99.38
7	490	3	7	98.00	99.39
8	479	3	18	95.80	99.38
9	472	5	23	94.40	99.95
总计	4844	33	123	96.88	99.39
比率%	96.88	0.66	2.46	96.88	99.39

本算法通过与直接用 0,1 点阵输入 BP 网络来识别的算法以及利用某些字符特征直接输入 BP 网络的算法相比,识别率有一定的提高,而且易于实现。

参考文献:

[1] 王 伟,盛立东.基于级连分组 BP 网络的高精度手写数字识别[J].中文信息学报,2000,14(2):61-62.

[2] 付庆铃,韩力群.基于人工神经网络的手写数字识别[J].北京工商大学学报:自然科学版,2004,122(13):44-45.

[3] Hagan M T,Demuth H B,Beale M H. Neural Network Design[M]. Beijing:China Machine Press,2002.

[4] 杜彦蕊.基于特征编码的手写字符识别技术.计算机工程,2004,30(5):156-157.

[5] 朱 江,宣国荣.一种基于骨架特征顺序编码的脱机手写体数字识别方法[J].小型微型计算机系统,2001,22(8):958-959.

(上接第 129 页)

3 函数调用与异常处理的异同

最后对比函数调用和异常处理之间的异同。throw 表达式的行为有点像函数的调用,而 catch 子句有点像函数定义。函数调用和异常处理的主要区别是:建立函数调用所需要的全部信息在编译时已经获得,而异常处理机制要求运行时的支持。对于普通函数调用,通过函数重载解析过程,编译器知道在调用点上哪个函数会真正被调用。但对于异常处理,编译器不知道特定的 throw 表达式的 catch 子句在哪个函数中,以及在处理异常之后执行权被转移到哪里。这些都在运行时刻决定,异常是随机发生的,处理异常的 catch 子句是沿调用链逆向进行查找,这与运行时的多态——虚函数也是不一样的。当一个异常不存在处理代码时,编译器无法通知用户,所以要有 terminate()函数,它是一种运行机制,当没有处理代码(catch 子句)能够匹配被抛出的异常时由它通知用户[5]。

4 结束语

程序在编写及运行时不可避免地会出现异常,使程序不能正常运行甚至崩溃,而 C++异常处理机制很好地解决了这一问题,使得程序更加简洁、健壮,同时也增强了程序的清晰性和可维护性。

参考文献:

[1] 裘宗燕.C++语言异常处理机制的研究[J].计算机科学,2003,30(11):155-156.

[2] Deitel H M,Deitel P J. Visual C++.NET 高级编程[M].郭凯,等译.北京:清华大学出版社,2004.

[3] 谭浩强.C++程序设计[M].北京:清华大学出版社,2004.

[4] 吴乃陵.C++程序设计[M].北京:高等教育出版社,2003.

[5] 海 燕,李晓玲.异常处理技术在 C++中的编程实现[J].计算机技术与自动化,2005,24(3):72-74.