第五讲 异常处理

一、异常简介

二、异常的使用

一、异常简介

在软件开发中，异常处理机制是一种比较有效的处理系统运行时错误的方法。C++针对异常处理提供了一种标准的方法，用于处理程序运行时的错误，保证软件系统运行的稳定性与健壮性。但是异常处理没有普通方法函数调用速度快。过度的错误处理会影响应用程序运行的效率。

使用异常，就把错误和处理分开来，由被调函数抛出异常，由调用者捕获这个异常，调用者就可以对错误进行处理。

二、异常的使用

异常的抛出和处理主要使用了三个关键字：try、throw和catch。其中抛出异常采用throw语句实现，其基本格式为:throw 表达式; ,将有可能会抛出异常的语句包围在try块中，一旦try块发现了异常，则这个异常可以被try语句块后的某个catch块捕获并处理，捕获和处理的条件是抛出的异常类型与catch块参数类型相匹配。由于C++使用数据类型来区分不同的异常，因此在判断异常时，throw语句中的表达式的类型就尤其重要。

try-catch块的形式如下:

```
try
{
    包含可能抛出异常的语句;
}
catch(类型名 [形参名])//捕获特定类型的异常
{
    处理异常的语句;
}
...
```

【例2-1】除数为0的常规处理方式

```
#include <iostream>
using namespace std;

bool func( float a, float b, float& c ) {
    if (b==0){
        return false;
    }
    c =  a/b;
    return true;
}

int main(int argc, const char * argv[]) {

    float a = 10, b = 0, c = 0;
    bool result = func( a, b, c );
    //根据函数返回的错误码，来判断是否有错误发生。
    if (!result){
        cout<<"The func fails!"<<endl;
        return 0;
    }else{
        cout<<"The func succeeds!"<<endl;
    }
    return 0;
}
```

程序运行结果如下：

```
The func fails!
```

【例2-2】处理除数为0的异常

```
#include <iostream>
using namespace std;
void func( float a, float b, float& c ){
    if (b==0){
        //当if条件成立，抛出异常，并且停止函数
        throw "Divided by zero";
    }
    c =  a/b;
}

int main(int argc, const char * argv[]) {
    float a = 10, b = 0, c = 0;
    try{
        //把有可能发生错误的函数或者代码放入try语句块中
        func( a, b, c );
    }
    catch( const char* str ){//异常处理：参数表示处理字符串类型异常
        cout<<str<<endl;
    }
    return 0;
}
```

程序运行结果如下：
Divided by zero

示例分析:

(1) 例2-1中为运行时错误的第一种处理方式，将错误代码放在了错误发生的地方，使其有很明显的错误处理，但是应用逻辑的错误处理混乱使其难以遵循应用逻辑;

(2) 例2-2中使用异常处理运行时错误，可以使应用逻辑更容易遵循和维护;

(3) 有一种可以捕捉到任何类型异常的块,如下所示:

```
catch( ... )
{
   cont<<“error happened!”<<endl;
}
```

这种catch块是一个“保底”办法，防止因为有些异常没有匹配的catch块，而导致程序异常终止。因此，一般将匹配程度高的catch块放在前面。而无参的catch块，通用性最强，针对性最差，应该把这个“通用”catch块放在所有catch块的后面。