



程序设计实习

郭炜 微博 <http://weibo.com/guoweiofpku>

<http://blog.sina.com.cn/u/3266490431>

刘家瑛 微博 <http://weibo.com/pkuliujiaying>



异常处理

异常处理

- C++异常处理基础：try、throw、catch
- 异常声明(exception specification)
- 意外异常(unexpected exception)
- 异常处理的作用
- 动态内存管理的异常处理
 - new

异常处理

- 程序运行中总难免发生错误
 - 数组元素的下标超界、访问NULL指针
 - 除数为0
 - 动态内存分配new需要的存储空间太大
 -
- 引起这些异常情况的原因：
 - 代码质量不高，存在BUG
 - 输入数据不符合要求
 - 程序的算法设计时考虑不周到
 -
- 我们总希望在发生异常情况时
 - 不只是简单地终止程序运行
 - 能够反馈异常情况的信息：哪一段代码发生的、什么异常
 - 能够对程序运行中已发生的事情做些处理：取消对输入文件的改动、释放已经申请的系统资源

- 通常的做法是：在预计会发生异常的地方，加入相应的代码，但这种做法并不总是适用的

```
.....//对文件A进行了相关的操作  
fun(arg, .....);//可能发生异常  
.....
```

- caller该如何知道fun(arg,)是否发生异常
 - 没有发生异常，可以继续执行
 - 发生异常，应该在结束程序运行前还原对文件A的操作
- fun(arg,)是别人已经开发好的代码
 - fun(arg,)的编写者不知道其他人会如何使用这个函数
 - fun(arg,)会出现在表达式中，通过返回值的方式区分是否发生异常
 - 不符合编写程序的习惯
 - 可能发生多种异常，通过返回值判断也很麻烦
- 需要一种手段
 - 把异常与函数的接口分开，并且能够区分不同的异常
 - 在函数体外捕获所发生的异常,并提供更多的异常信息

用try、catch处理异常

```
#include <iostream>
using namespace std;
int main()
{
    double m ,n;
    cin >> m >> n;
    try {
        cout << "before dividing." << endl;
        if( n == 0)
            throw -1; //抛出int类型异常
        else
            cout << m / n << endl;
        cout << "after dividing." << endl;
    }
```

```
catch(double d) {  
    cout << "catch(double) " << d << endl;  
}  
catch(int e) {  
    cout << "catch(int) " << e << endl;  
}  
cout << "finished" << endl;  
return 0;  
}
```

程序运行结果如下：

9.6

before dividing.

1.5

after dividing.

finished

//捕获任何异常的catch块

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    double m ,n;
```

```
    cin >> m >> n;
```

```
    try {
```

```
        cout << "before dividing." << endl;
```

```
        if( n == 0)
```

```
            throw -1; //抛出整型异常
```

```
        else if( m == 0 )
```

```
            throw -1.0; //抛出double型异常
```

```
        else
```

```
            cout << m / n << endl;
```

```
        cout << "after dividing." << endl;
```

```
    }
```



```
    catch(double d) {  
        cout << "catch(double) " << d << endl;  
    }  
    catch(...) {  
        cout << "catch(...)" << endl;  
    }  
    cout << "finished" << endl;  
    return 0;  
}
```

程序运行结果:

9 0 ✓

before dividing.

catch(...)

finished

0 6 ✓

before dividing.

catch(double) -1

finished

异常的再抛出

如果一个函数在执行的过程中，抛出的异常在本函数内就被catch块捕获并处理了，那么该异常就不会抛给这个函数的调用者(也称“上一层的函数”)；如果异常在本函数中没被处理，就会被抛给上一层的函数。

//异常再抛出

```
#include <iostream>
#include <string>
using namespace std;
class CException
{
    public :
        string msg;
        CException(string s):msg(s) { }
};
```

```
double Devide(double x, double y)
{
    if(y == 0)
        throw CException("devided by zero");
    cout << "in Devide" << endl;
    return x / y;
}

int CountTax(int salary)
{
    try {
        if( salary < 0 )
            throw -1;
        cout << "counting tax" << endl;
    }
    catch (int ) {
        cout << "salary < 0" << endl;
    }
}
```

```

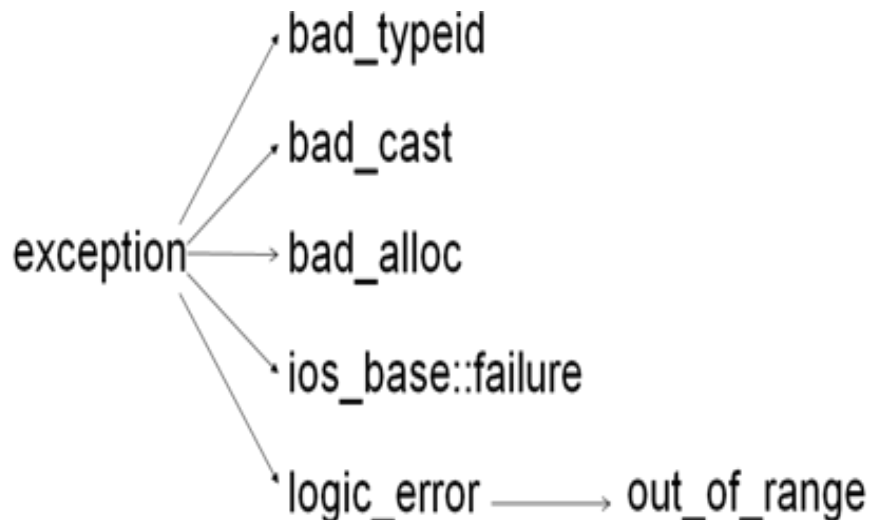
    cout << "tax counted" << endl;
    return salary * 0.15;
}
int main()
{
    double f = 1.2;
    try {
        CountTax(-1);
        f = Devide(3,0);
        cout << "end of try block" << endl;
    }
    catch(CException e) {
        cout << e.msg << endl;
    }
    cout << "f=" << f << endl;
    cout << "finished" << endl;
    return 0;
}

```

输出结果:
salary < 0
tax counted
devided by zero
f=1.2
finished

C++标准异常类

- C++标准库中有一些类代表异常，这些类都是从exception类派生而来。常用的几个异常类如下：



bad_cast

在用 dynamic_cast 进行从多态基类对象（或引用），到派生类的引用的强制类型转换时，如果转换是不安全的，则会抛出此异常。

```
#include <iostream>
#include <stdexcept>
#include <typeinfo>
using namespace std;
class Base
{
    virtual void func(){}
};
class Derived : public Base
{
public:
    void Print() { }
};
```

```
void PrintObj( Base & b)
{
    try {
        Derived & rd = dynamic_cast<Derived&>(b);
        //此转换若不安全，会抛出bad_cast异常
        rd.Print();
    }
    catch (bad_cast& e)      {
        cerr << e.what() << endl;
    }
}

int main ()
{
    Base b;
    PrintObj(b);
    return 0;
}
```

输出结果：
Bad dynamic_cast!

bad_alloc

在用new运算符进行动态内存分配时，如果没有足够的内存，则会引发此异常。

```
#include <iostream>
#include <stdexcept>
using namespace std;
int main ()
{
    try {
        char * p = new char[0x7fffffff]; //无法分配这么多空间，会抛出异常
    }
    catch (bad_alloc & e) {
        cerr << e.what() << endl;
    }
    return 0;
}
```

输出结果：
bad allocation

out_of_range

用vector或string的at成员函数根据下标访问元素时，如果下标越界，就会抛出此异常。例如：

```
#include <iostream>
#include <stdexcept>
#include <vector>
#include <string>
using namespace std;
int main ()
{
    vector<int> v(10);
    try {
        v.at(100)=100; //抛出out_of_range异常
    }
    catch (out_of_range& e) {
        cerr << e.what() << endl;
    }
}
```

```
string s = "hello";
try {
    char c = s.at(100); //抛出out_of_range异常
}
catch (out_of_range& e) {
    cerr << e.what() << endl;
}
return 0;
}
```

输出结果:

invalid vector<T> subscript
invalid string position

In-Video Quiz

下面哪种说法是正确的？

- A) 一个函数抛出异常后，必须在函数内部处理该异常，否则程序就会中止
- B) 只要写了catch块，那么在try块中抛出的异常，一定会被某个catch块捕获并处理
- C) 程序中抛出的异常如果没有被任何catch块处理，则会导致程序中止
- D) try块中抛出异常后，如果该异常被catch块捕获并处理，处理后就会继续执行try块中的语句。

In-Video Quiz

下面哪种说法是正确的？

- A) 一个函数抛出异常后，必须在函数内部处理该异常，否则程序就会中止
- B) 只要写了catch块，那么在try块中抛出的异常，一定会被某个catch块捕获并处理
- C) 程序中抛出的异常如果没有被任何catch块处理，则会导致程序中止
- D) try块中抛出异常后，如果该异常被catch块捕获并处理，处理后就会继续执行try块中的语句。

#C