第十二章 异常处理

清华大学　郑　莉

导学

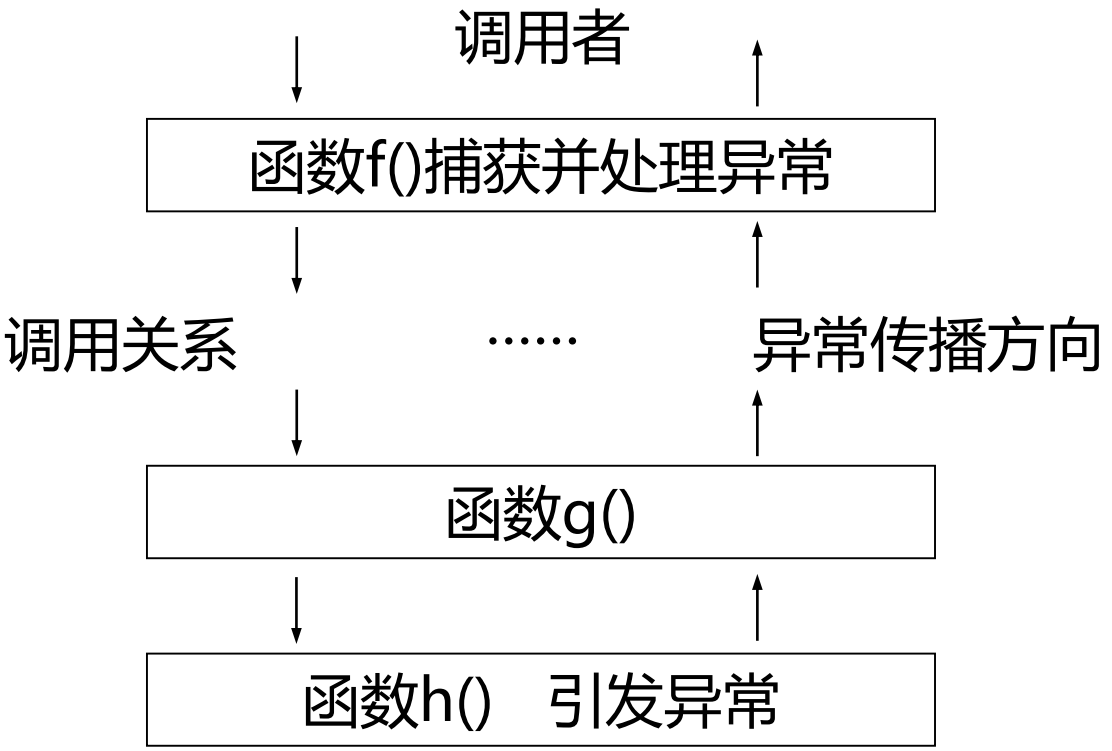
- 我们在编写应用软件时，不仅要保证软件的正确性，而且应该具有容错能力。
  - 也就是说，不仅在正确的环境条件下、在用户正确操作时要运行正确，而且在环境条件出现意外或用户使用操作不当的情况下，也应该有正确合理的表现，不能轻易出现死机，更不能出现灾难性的后果。
- 由于环境条件和用户操作的正确性是没有百分之百保障的，所以我们在设计程序时，就要充分考虑到各种意外情况，并给与恰当的处理。这就是我们所说的异常处理。

目录

异常处理的思想与程序实现

<12.1~12.2>

12.1 异常处理的基本思想



12.2.1 异常处理的语法

- 抛掷异常的程序段

```
……
throw 表达式;
……
```

- 捕获并处理异常的程序段

```
try
   复合语句                    保护段
catch（异常声明）
   复合语句                    异常处理程序
catch（异常声明）
   复合语句
   …
```

- 若有异常则通过throw创建一个异常对象并抛掷
- 将可能抛出异常的程序段嵌在try块之中。通过正常的顺序执行到达try语句，然后执行try块内的保护段
- 如果在保护段执行期间没有引起异常，那么跟在try块后的catch子句就不执行。程序从try块后的最后一个catch子句后面的语句继续执行
- catch子句按其在try块后出现的顺序被检查。匹配的catch子句将捕获并处理异常（或继续抛掷异常）。
- 如果匹配的处理器未找到，则库函数terminate将被自动调用，其缺默认能是调用abort终止程序。

例12-1处理除零异常

例12-1处理除零异常

```
//12_1.cpp
#include <iostream>
using namespace std;
int divide(int x, int y) {
    if (y == 0)
        throw x;
    return x / y;
}
int main() {
    try {
        cout << "5 / 2 = " << divide(5, 2) << endl;
        cout << "8 / 0 = " << divide(8, 0) << endl;
        cout << "7 / 1 = " << divide(7, 1) << endl;
    } catch (int e) {
        cout << e << " is divided by zero!" << endl;
    }
    cout << "That is ok." << endl;
    return 0;
}
```

运行结果：

```
5 / 2 = 2
8 is divided by zero!
That is ok.
```

异常接口声明

- 一个函数显式声明可能抛出的异常，有利于函数的调用者为异常处理做好准备

异常接口声明

- 可以在函数的声明中列出这个函数可能抛掷的所有异常类型。
  - 例如：
    ```
    void fun() throw(A , B , C , D);
    ```
- 若无异常接口声明，则此函数可以抛掷任何类型的异常。
- 不抛掷任何类型异常的函数声明如下：

```
void fun() throw();
```

异常处理中的构造与析构

<12.3>

自动的析构

- 找到一个匹配的catch异常处理后
  - 初始化异常参数。
  - 将从对应的try块开始到异常被抛掷处之间构造（且尚未析构）的所有自动对象进行析构。
  - 从最后一个catch处理之后开始恢复执行。

例12-2 带析构语义的类的C++异常处理

例12-2 带析构语义的类的C++异常处理

```
//12_2.cpp
#include <iostream>
#include <string>
using namespace std;
class MyException {
public:
    MyException(const string &message) : message(message) {}
    ~MyException() {}
    const string &getMessage() const { return message; }
private:
    string message;
};

class Demo {
public:
    Demo() { cout << "Constructor of Demo" << endl; }
    ~Demo() { cout << "Destructor of Demo" << endl; }
};
```

例12-2 带析构语义的类的C++异常处理

```
void func() throw (MyException) {
    Demo d;
    cout << "Throw MyException in func()" << endl;
    throw MyException("exception thrown by func()");
}

int main() {
    cout << "In main function" << endl;
    try {
            func();
    } catch (MyException& e) {
            cout << "Caught an exception: " << e.getMessage() << endl;
    }
    cout << "Resume the execution of main()" << endl;
    return 0;
}
```

例12-2 带析构语义的类的C++异常处理

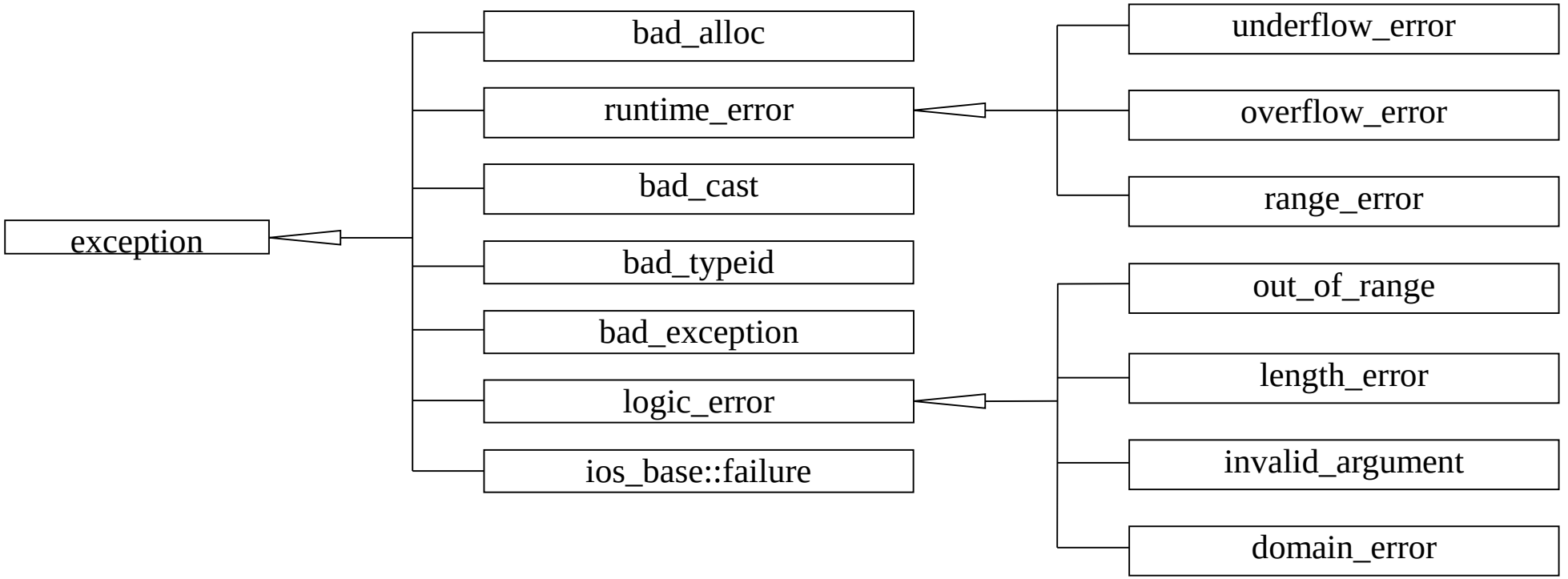
运行结果：

```
In main function
Constructor of Demo
Throw MyException in func()
Destructor of Demo
Caught an exception: exception thrown by func()
Resume the execution of main()
```

标准程序库异常处理

< 12.4 >

标准异常类的继承关系


exception
bad_alloc
runtime_error
bad_cast
bad_typeid
bad_exception
logic_error
ios_base::failure
underflow_error
overflow_error
range_error
out_of_range
length_error
invalid_argument
domain_error

C++标准库各种异常类所代表的异常

异常类	头文件	异常的含义
bad_alloc	exception	用new动态分配空间失败
bad_cast	new	执行dynamic_cast失败（dynamic_cast参见8.7.2节）
bad_typeid	typeinfo	对某个空指针p执行typeid(*p)（typeid参见8.7.2节）
bad_exception	typeinfo	当某个函数fun()因在执行过程中抛出了异常声明所不允许的异常而调用unexpected()函数时，若unexpected()函数又一次抛出了fun()的异常声明所不允许的异常，且fun()的异常声明列表中有bad_exception，则会有一个bad_exception异常在fun()的调用点被抛出
ios_base::failure	ios	用来表示C++的输入输出流执行过程中发生的错误
underflow_error	stdexcept	算术运算时向下溢出
overflow_error	stdexcept	算术运算时向上溢出
range_error	stdexcept	内部计算时发生作用域的错误
out_of_range	stdexcept	表示一个参数值不在允许的范围之内
length_error	stdexcept	尝试创建一个长度超过最大允许值的对象
invalid_argument	stdexcept	表示向函数传入无效参数
domain_error	stdexcept	执行一段程序所需要的先决条件不满足

标准异常类的基础

- 其中有三个类是标准异常类的基础

标准异常类的基础

- exception：标准程序库异常类的公共基类
- logic_error表示可以在程序中被预先检测到的异常
  - 如果小心地编写程序，这类异常能够避免
- runtime_error表示难以被预先检测的异常

例12-3 三角形面积计算

例12-3 三角形面积计算

- 编写一个计算三角形面积的函数，函数的参数为三角形三边边长*a*、*b*、*c*，可以用Heron公式计算：
- 设 $p=\frac{a+b+c}{2}$，则三角形面积 $S=\sqrt{p(p-a)(p-b)(p-c)}$

例12-3 三角形面积计算

```
//12_3.cpp
#include <iostream>
#include <cmath>
#include <stdexcept>
using namespace std;
//给出三角形三边长，计算三角形面积
double area(double a, double b, double c)  throw (invalid_argument)
{
   //判断三角形边长是否为正
   if (a <= 0 || b <= 0 || c <= 0)
       throw invalid_argument("the side length should be positive");
   //判断三边长是否满足三角不等式
   if (a + b <= c || b + c <= a || c + a <= b)
       throw invalid_argument("the side length should fit the triangle inequation");
   //由Heron公式计算三角形面积
   double s = (a + b + c) / 2;
   return sqrt(s * (s - a) * (s - b) * (s - c));
}
```

例12-3 三角形面积计算

```
int main() {
    double a, b, c;    //三角形三边长
    cout << "Please input the side lengths of a triangle: ";
    cin >> a >> b >> c;
    try {
        double s = area(a, b, c);   //尝试计算三角形面积
        cout << "Area: " << s << endl;
    } catch (exception &e) {
        cout << "Error: " << e.what() << endl;
    }
    return 0;
}
```

例12-3 三角形面积计算

- 运行结果1：

Please input the side lengths of a triangle: 3 4 5
Area: 6

- 运行结果2：

Please input the side lengths of a triangle: 0 5 5
Error: the side length should be positive

- 运行结果2：

Please input the side lengths of a triangle: 1 2 4
Error: the side length should fit the triangle inequation

12.7 小结

- 主要内容
  - 异常处理的基本思想、C++异常处理的实现、异常处理中的构造与析构
- 达到的目标
  - 简单了解C++的异常处理机制

小结讲稿

- 程序运行中的有些错误是可以预料但不可避免的，当出现错误时，要力争做到允许用户排除环境错误，继续运行程序，这就是异常处理程序的任务。C++语言提供对处理异常情况的内部支持。try、throw和catch语句就是C++语言中用于实现实现异常处理的机制。
- 为了加强程序的可读性，使函数的用户能够方便地知道所使用的函数会抛掷哪些异常，可以在函数的声明中列出这个函数可能抛掷的所有异常类型，这就是异常接口声明。
- C++异常处理的真正能力，不仅在于它能够处理各种不同类型的异常，还在于它具有在堆栈展开期间为异常抛掷前构造的所有局部对象自动调用析构函数的能力。
- 最后，对C++标准程序库中标准异常类及功能进行了介绍，同时介绍了C++标准程序库对异常处理作的保证，即C++标准程序库在面对异常时，应当保证不会发生资源泄漏，也不能破坏容器的不变特性。