Chapter 16

Exception Handling

http://jssec.seu.edu.cn

yangming2002@seu.edu.cn
What exceptions are and when to use them.

To use **try**, **catch** and **throw** to detect, handle and indicate exceptions, respectively.

To process uncaught and unexpected exceptions.

To declare new exception classes.

How stack unwinding enables exceptions not caught in one scope to be caught in another scope.

To handle **new** failures.

To understand the standard exception hierarchy.
16.1 Introduction

16.2 Scenario A: Handle exception thrown by C++ standard lib

16.3 Scenario B: Define, throw and handle your own exception

16.4 Stack Unwinding
16.1 Introduction

- Exception (异常): An exception is an indication of a problem that occurs during a program's execution. 程序执行期间, 可检测到的不正常情况。

- Examples: 0作除数; 数组下标越界; 打开不存在的文件; 内存分配失败
16.1 Introduction

- Intermixing program logic
  
  Perform a task
  If the preceding task did not execute correctly
  Perform error processing

  Perform next task
  If the preceding task did not execute correctly
  Perform error processing

  ... ... ...

- Difficult to read, modify, maintain and debug—especially in large applications
- Low performance
16.1 Introduction

- **Exception handling (异常处理)**: In many cases, handling an exception allows a program to continue executing as if no problem had been encountered.

![Diagram showing exception handling](image)

- **try-catch**: robust (健壮性) and fault-tolerant (容错)
16.1 Introduction

- How to define our own Exception?
- How to throw Exception?
- How to catch and handle Exception?
- Stack Unwinding (栈展开机制)

- **Scenario A**: Handle exception thrown by C++ standard lib.
- **Scenario B**: Define, throw and handle your own exception.
16.1 Introduction
16.2 Scenario A: Handle exception thrown by C++ standard lib
16.3 Scenario B: Define, throw and handle your own exception
16.4 Stack Unwinding
16.2 Scenario A: Handle exception thrown by C++ standard lib

- 需求：如何处理C++库调用时抛出的异常？
- **try-catch**语句

Termination Model of Exception Handling

P350 Fig.11.1
Attempt to assign 'd' to s1.at( 30 ) yields:

This application has requested the Runtime to terminate it in an unusual way. Please contact the application's support team for more information.
16.2 Scenario A: Handle exception thrown by C++ standard lib

```cpp
1. class Test{
2. public:
3.    Test(){ cout << "Constructor called." << endl; }
4.    ~Test(){ cout << "Destructor ok." << endl; }
5. }
6. int main()
7. {
8.    Test t;
9.    double *ptr[ 50 ];
10.   
11.   for ( int i = 0; i < 50; i++ )
12.   {
13.      ptr[ i ] = new double[ 50000000 ];
14.      cout << "Allocated 50000000 doubles in ptr[ " << i << " ]\n";
15.   }
16.   return 0;
17. }
```
由于 `new` 操作失败，程序 `abort`。

危害：剩余对象全部不析构等

```
Constructor called.
Allocated 50000000 doubles in ptr[ 0 ]
Allocated 50000000 doubles in ptr[ 1 ]
Allocated 50000000 doubles in ptr[ 2 ]
Allocated 50000000 doubles in ptr[ 3 ]
Allocated 50000000 doubles in ptr[ 4 ]
```

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
If `new` fails to allocate memory and `set_new_handler` did not register a new-handler function, `new` throws a `bad_alloc` exception [From MSDN].

- **Choice 1**: Register a new-handler function
- **Choice 2**: Handle `bad_alloc` exception
16.2 Scenario A: Handle exception thrown by C++ standard lib

- Exception handler典型操作:
  - reports the error to the user,
  - logs it to a file
  - terminates the program gracefully
  - tries an alternate strategy to accomplish the failed task
### 16.2 Scenario A: Handle exception thrown by C++ standard lib

```cpp
1. try {
2. // code that may throw exceptions
3. }
4. catch (exception-declaration) {
5. // code that executes when
6. // exception-declaration is thrown
7. }
8. catch (exception-declaration) {
9. // code that handles another exception type
10. }
11. catch (exception-declaration) {
12. // 定异常类型变量的声明，如: catch(bad_alloc& theexception)
13. // 通常是异常类型的引用
14. // 如要捕捉所有的异常，则: catch( … )
```
16.2 Scenario A: Handle exception thrown by C++ standard lib

Termination Model of Exception Handling

1. 抛出异常时，try block结束执行；
2. 寻找匹配的catch handler (is-a);
3. 执行catch handler代码；
4. 程序控制跳至最后一个catch handler后的首条语句。（注意：不再执行try block中抛出异常点的后续语句）
16.2 Scenario A: Handle exception thrown by C++ standard lib

1. `try` {
2.     // code that may throw exceptions
3. } 抛出异常, skip try中的后续语句, 程序控制转至catch语句
4. `catch (exception-declaration)` {
5.     // code that executes when
6.     // exception-declaration is thrown
7. } 若未匹配is-a, 转至下一条catch语句
8. `catch (exception-declaration)` {
9.     // code that handles another exception
10. } 若匹配, 执行异常处理代码
11. `catch (exception-declaration)` {
12. } 跳过剩余的catch, 执行后续的代码
13. `cout << “following statements”;`
16.2 Scenario A: Handle exception thrown by C++ standard lib

```cpp
int main()
{
    Test t;
    double *ptr[ 50 ];

    try
    {
        for ( int i = 0; i < 50; i++ )
        {
            ptr[ i ] = new double[ 50000000 ]; // may throw exception
            cout << "Allocated 50000000 doubles in ptr[ " << i << " ]\n";
        }
    }

    catch ( bad_alloc &memoryAllocationException ) // handle exception
    {
        cerr << "Exception occurred: "
             << memoryAllocationException.what() << endl;
    }

    cout << "Exception handled." << endl;
    return 0;
}
```

Constructor called.
Allocated 50000000 doubles in ptr[ 0 ]
Allocated 50000000 doubles in ptr[ 1 ]
Allocated 50000000 doubles in ptr[ 2 ]
Allocated 50000000 doubles in ptr[ 3 ]

Exception occurred: bad allocation
Exception handled.
Destructor ok.

exception类定义的虚函数, returns error message.
16.2 Scenario A: Handle exception thrown by C++ standard lib

- 修改1: bad_alloc → exception
- 修改2: bad_alloc → logic_error
- 修改3: bad_alloc → …

Figure 16.11. Standard Library exception classes
is-a: A match occurs if the types are identical or if the thrown exception's type is a derived class of the exception-parameter type.

如果没有catch匹配, 见课件16.4节
16.1 Introduction
16.2 Scenario A: Handle exception thrown by C++ standard lib
16.3 Scenario B: Define, throw and handle your own exception
16.4 Stack Unwinding
如何在自己的函数中抛出异常？
需求：设计quotient函数，对用户输入的两个数进行除法操作，希望输入的除数为0时能抛出异常，由调用函数捕获并处理该异常

Exception Specifications 异常说明
16.3 Scenario B: Define, throw and handle your own exception

// P489. Figure 16.1. Class DivideByZeroException definition
3. #include <stdexcept>
4. using std::runtime_error;
5.
6. class DivideByZeroException : public runtime_error
7. {
8.   public:
9.     DivideByZeroException()
10.      : runtime_error( "attempted to divide by zero" ) {}
11.   };

// P490. Figure 16.2. throws and handle exceptions
13. double quotient( int numerator, int denominator )
14. {
15.   if ( denominator == 0 )
16.     throw DivideByZeroException(); // terminate function
17.   return static_cast< double >( numerator ) / denominator;
18. }
16.3 Scenario B: Define, throw and handle your own exception

```cpp
36. try
37. {
38.     result = quotient( number1, number2 );
39.     cout << "The quotient is: " << result << endl;
40. } // end try
41. catch ( DivideByZeroException &divideByZeroException )
42. {
43.     cout << "Exception occurred: " << divideByZeroException.what() << endl;
44. }
45. } // end catch
46.
47. cout << "\nEnter two integers (end-of-file to end): ";
```

Enter two integers (end-of-file to end): 10 6
The quotient is: 1.66667

Enter two integers (end-of-file to end): 10 0
Exception occurred: attempted to divide by zero
16.3 Scenario B: Define, throw and handle your own exception

```cpp
int someFunction( double value )
throw (ExceptionA, ExceptionB, ExceptionC )
{
    // function body
}
```

- **Exception specification** 异常说明：列出函数可抛出的异常
  - 函数可抛出指定异常或其派生类型
  - 当函数抛出异常不在异常说明的列表中，自动调用C++标准库 `unexpected` 函数(缺省将调用 `terminate` 函数)
  - 如果 `throw` 后为空括号( ), 表示不得抛出任何异常，此时如果函数仍然抛出异常，同样由 `unexpected` 处理

- 如果不给出异常说明，函数可以抛出任何异常
16.1 Introduction
16.2 Scenario A: Handle exception thrown by C++ standard lib
16.3 Scenario B: Define, throw and handle your own exception
16.4 Stack Unwinding
16.4 Stack Unwinding

Stack Unwinding (栈展开机制)

1. 当某个代码块 (异常源) 抛出异常，将立即结束该代码块的正常执行，根据函数调用链 (首先是本函数) 回溯寻找可以 catch 该异常的 Handler；

2. 如果找到了匹配的 Handler，则执行 Stack Unwinding，即：逆序释放进入 Handler 所在函数的 try 语句后构造的所有局部对象 (异常源 → Handler)；

3. 如果回溯到 main 函数，仍没有找到匹配的 Handler，则调用 terminate 函数（该函数缺省调用 abort，不执行栈展开），结束程序。
1. void function3() throw (runtime_error)  
2. {  
3.  cout << "In fun3\n";  
4.  Test t(3);  
5.  throw runtime_error("runtime_error in fun3");  
6.  
7.  cout << "Reach here? fun3\n";  
8. }  

1. void function2() throw (runtime_error)  
2. {  
3.  Test t(2);  
4.  cout << "fun3 is called inside fun2\n";  
5.  function3();  
6.  cout << "Reach here? fun2\n";  
7. }  

1. void function1() throw (runtime_error)  
2. {  
3.  Test t(1);  
4.  cout << "fun2 is called inside fun1\n";  
5.  function2();  
6.  cout << "Reach here? Fun1\n";  
7. }  

1. int main()  
2. {  
3.  try {  
4.  cout << "fun1 is called inside main\n";  
5.  function1();  
6.  cout << "Reach here? fun main\n";  
7.  }  
8.  catch (runtime_error &error) {  
9.  cout << "Exception occurred: "  
10. << error.what() << endl;  
11.  cout << "Exception handled in main\n";  
12.  }  
13.  return 0;  
14. }  

fun1 is called inside main  
Constructor 1  
fun2 is called inside fun1  
Constructor 2  
fun3 is called inside fun2  
In fun3  
Constructor 3
16.4 Stack Unwinding

```cpp
1. void function3() throw ( runtime_error )
2. {
3.    cout << "In fun3\n";
4.    Test t(3);
5.    throw runtime_error( "runtime_error in fun3" );
6.    cout << "Reach here? fun3\n";
7.}

1. void function2() throw ( runtime_error )
2. {
3.    Test t(2);
4.    cout << "fun3 is called inside fun2\n";
5.    function3();
6.    cout << "Reach here? fun2\n";
7.}

1. void function1() throw ( runtime_error )
2. {
3.    Test t(1);
4.    cout << "fun2 is called inside fun1\n";
5.    function2();
6.    cout << "Reach here? fun1\n";
7.}

1. int main()
2. {
3.    try {
4.        cout << "fun1 is called inside main\n";
5.        function1();
6.        cout << "Reach here? fun main\n";
7.    } catch ( runtime_error &error ) {
8.        cout << "Exception occurred: "
9.        << error.what() << endl;
10.       cout << "Exception handled in main\n";
11.    }
12.    return 0;
13.}
```

Destructor 3
Destructor 2
Destructor 1
Exception occurred: runtime_error
in fun3
Exception handled in main

Stack unwinding occur
### 16.4 Stack Unwinding

**void function3() throw (runtime_error)**
1. {
2.    
3.    cout << "In fun3\n";
4.    Test t(3);
5.    throw runtime_error("runtime_error in fun3");
6.    cout << "Reach here? fun3\n";
7.}

**void function2() throw (runtime_error)**
1. {
2.    
3.    Test t(2);
4.    cout << "fun3 is called inside fun2\n";
5.    function3();
6.    cout << "Reach here? fun2\n";
7.}

**void function1() throw (runtime_error)**
1. {
2.    
3.    Test t(1);
4.    cout << "fun2 is called inside fun1\n";
5.    function2();
6.    cout << "Reach here? fun1\n";
7.}

**int main()**
1. {
2.    
3.    try {
4.        cout << "fun1 is called inside main\n";
5.        function1();
6.        cout << "Reach here? fun main\n";
7.    }
8.    catch (runtime_error &error) {
9.        cout << "Exception occurred: " << error.what() << endl;
10.       cout << "Exception handled in main\n";
11.    }
12.    return 0;
13.}

---

**This application has requested the Runtime to terminate it in an unusual way.**
Please contact the application's support team for more information.

- **fun1 is called inside main**
- **Constructor 1**
- **fun2 is called inside fun1**
- **Constructor 2**
- **fun3 is called inside fun2**
- **In fun3**
- **Constructor 3**

**logic_error**
16.4 Stack Unwinding

- 如果**Exception Handler**无法处理捕获的异常，可以**re-throw**重新抛出异常：

  ```
  throw;
  ```
1. class Obj{
2.     int id;
3. public:
4.     Obj(int n){
5.         id = n;
6.         cout << "ctor " << id << endl;
7.     }
8.     ~Obj(){
9.         cout << "dtor " << id << endl;
10.    }
11.};
12. void f2(){
13.    Obj o(2);
14.    double a = 0;
15.    try{
16.        throw a;
17.    }
18.    catch(double){
19.        cout << "OK2!" << endl;
20.        throw;
21.    }
22.    cout << "end2" << endl;
23.}
24. void f1(){
25.    Obj o(1);
26.    try{
27.        f2();
28.    }
29.    catch(char){
30.        cout << "OK1!" << endl;
31.    }
32.    cout << "end1" << endl;
33.}
34. int main(){
35.    try{
36.        Obj o(0);
37.        f1();
38.    }
39.    catch(double){
40.        cout << "OK0!" << endl;
41.    }
42.    cout << "end0" << endl;
43.    return 0;
44.}
异常的概念

try-throw-catch模块的语法和处理流程

栈展开过程（与构造和析构的关系）

new异常的处理
实验必选题目 (交实验报告):
16.26, 16.30

实验任选题目 (不交实验报告):

作业题目 (Homework):