第十七章 异常处理

异常处理 ⇒ 在程序运行过程中产生异常情况

比如尝试除以0的操作

C++ 的异常处理涉及三个关键字：try、catch、throw

- throw：当问题出现时，程序会依托 throw 块 抛出异常
- catch：在想要处理问题的地方，通过 catch 块捕获处理异常
- try：try 块中的代码将进行执行，并标识被激活的特定异常。

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

double division(int a, int b)
{
   if( b == 0 )
   {
      throw "Division by zero condition!";
   }
   return (a/b);
}

int main ()
{
   int x = 50;
   int y = 0;
   double z = 0;

   try {
     z = division(x, y);
     cout << z << endl;
   }catch (const char* msg) {
     cerr << msg << endl;
   }

   return 0;
}
```

抛出异常类型 const char*

保护代码，可能产生异常

捕获&处理异常

cerr 为ostream类预定对象，用于输出错误信息

a、抛出异常

开发者可以使用throw语句在代码块中的任何地方抛出异常

通常在函数体内，辅以选择等判断

开发者可以指定函数可能抛出的异常类型

通过给定异常规格说明；

开发者若不指定[不写throw]

则函数可以抛出任意类型的异常

若写throw()[参数空]

则函数不抛出异常

```
void fun(int n) throw (int, MyException, MyExceptionD)
{
    if (n == 1)
    {
        throw 1;
    }
    else if (n == 2)
    {
        throw MyException("test Exception");
    }
    else if (n == 3)
    {
        throw MyExceptionD("test ExceptionD");
    }

}
```

异常规格说明

b、捕获异常

catch块跟在try块后，用于捕获异常。

开发者可以指定想要捕捉的异常类型

⇓

由catch关键字后的括号内的异常声明决定

```
try
{
   // 保护代码
}catch( ExceptionName e )
{
  // 处理 ExceptionName 异常的代码
}
```

※ try块后一般罗列着多个 catch 块来处理不同异常

```
try
{
   // 保护代码
}catch( ExceptionName e1 )
{
   // catch 块
}catch( ExceptionName e2 )
{
   // catch 块
}catch( ExceptionName eN )
{
   // catch 块
}
```

可能出现异常的代码
通常为函数

异常类型

根据抛出的不同异常[类型]
进行不同处理

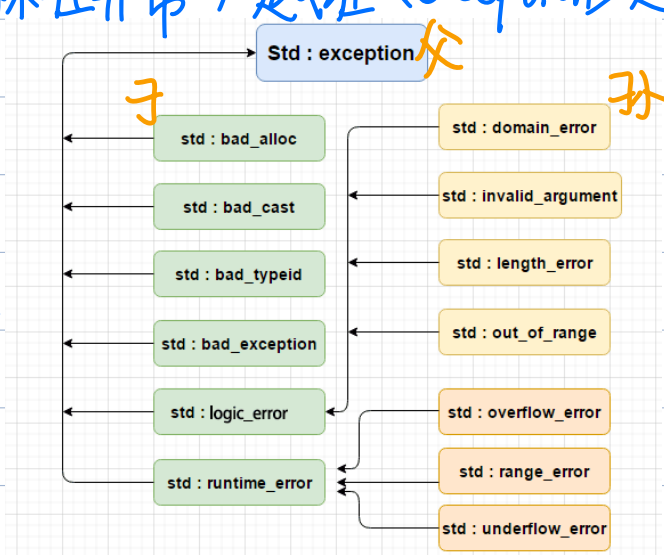
若希望让catch块能够处理任意异常
则异常声明括号内指定(...)

```
try
{
   // 保护代码
}catch(...)
{
  // 能处理任何异常的代码
}
```

C、C++中的已定义的标准异常

C++提供了一系列的标准异常，定义在<exception>类中，开发者可以直接调用。

这些异常是以父子类层次结构组织起来的。



下表是对上面层次结构中出现的每个异常的说明：

异常	描述
std::exception	该异常是所有标准 C++ 异常的父类。
std::bad_alloc	该异常可以通过 **new** 抛出。
std::bad_cast	该异常可以通过 **dynamic_cast** 抛出。
std::bad_typeid	该异常可以通过 **typeid** 抛出。
std::bad_exception	这在处理 C++ 程序中无法预期的异常时非常有用。
std::logic_error	理论上可以通过读取代码来检测到的异常。
std::domain_error	当使用了一个无效的数学域时，会抛出该异常。
std::invalid_argument	当使用了无效的参数时，会抛出该异常。
std::length_error	当创建了太长的 std::string 时，会抛出该异常。
std::out_of_range	该异常可以通过方法抛出，例如 std::vector 和 std::bitset<>::operator[]()。
std::runtime_error	理论上不可以通过读取代码来检测到的异常。
std::overflow_error	当发生数学上溢时，会抛出该异常。
std::range_error	当尝试存储超出范围的值时，会抛出该异常。
std::underflow_error	当发生数学下溢时，会抛出该异常。

※开发者可以通过继承<exception>类来定义新的异常，并通过覆盖<exception>类内的虚函数what()来返回异常原因.

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

struct MyException : public exception
{
  const char * what () const throw ()
  {
    return "C++ Exception";
  }
};

int main()
{
  try
  {
    throw MyException();
  }
  catch(MyException& e)
  {
    std::cout << "MyException caught" << std::endl;
    std::cout << e.what() << std::endl;
  }
  catch(std::exception& e)
  {
    //其他的错误
  }
}
```

继承<exception类>

```cpp
const char * what () const throw ()
{
//函数体
}
```

做几点说明，从左到右看：

1. **const char *** 表示返回值类型
2. **what** 是函数名称
3. **()** 是参数列表
4. **const** 表示该成员函数不能修改成员变量
5. **throw()** 是异常规格说明符。括号内写该函数可抛出的异常类型

上述 5 点均为函数的声明部分。