C++异常

1. 异常的语法

1.1 捕获全部的异常

```
try
  {
    // 可能抛出异常的代码。
    // throw 异常对象;
  }
  catch (...)
  {
    // 不管什么异常，都在这里统一处理。
  }
```

1.2 捕获指定的异常

```
  try
  {
    // 可能抛出异常的代码。
    // throw 异常对象;
  }
  catch (exception1 e)
  {
    // 发生exception1异常时的处理代码。
  }
  catch (exception2 e)
  {
    // 发生exception2异常时的处理代码。
  }
```

在`try`语句块中，如果没有发生异常，执行完`try`语句块中的代码后，将继续执行`try`语句块之后的代码；如果发生了异常，用`throw`抛出异常对象，异常对象的类型决定了应该匹配到哪个`catch`语句块，如果没有匹配到`catch`语句块，程序将调用`abort()`函数中止。

如果 `try` 语句块中用 `throw` 抛出异常对象，并且匹配到了 `catch` 语句块，执行完 `catch` 语句块中的代码后，将继续执行 `catch` 语句块之后的代码，不会回到 `try` 语句块中。

如果程序中的异常没有被捕获(没有被处理)，程序将异常中止。

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


int main(void)
{
    try
    {
        // 可能抛出的异常
        int ii = 0;
        cout << "你是一只什么鸟?(1-傻傻鸟  2-小小鸟): ";
        cin >> ii;

        if (ii == 1) {  // throw抛出const char* 类型的异常
            throw "不好，有人说我是一只傻傻鸟";
        }

        if (ii == 2)    // throw抛出int类型的异常
        {
            throw ii;
        }

        if (ii == 3)    // throw抛出string类型的异常
        {
            throw string("不好，有人说我是一只傻傻鸟");
        }


        cout << "我不是一只傻傻鸟" << endl;
    }

    catch (int ii)
    {
        cout << "异常的类型为int=" << ii << endl;
    }

    catch (const char* ss)
    {
        cout << "异常的类型为const char*=" << ss << endl;
    }

    catch (string str)
    {
        cout << "异常的类型为string=" << str << endl;
    }

    //catch (...)       // 不管什么异常，都在这处理
    //{
    //  cout << "捕获到异常，具体没管是什么异常" << endl;
    //}

    cout << "程序继续运行..." << endl;    // 执行完try ... catch...后，将继续执行程序中其他的代码
    return 0;
}
```

2. 异常的规范

C++98标准提出了异常规范，目的是为了让使用者知道函数可能会引发哪些异常。

```
void func1() throw(A, B, C);    // 表示该函数可能会抛出A、B、C类型的异常。

void func2() throw();       // 表示该函数不会抛出异常。

void func3();           // 该函数不符合C++98的异常规范。
```

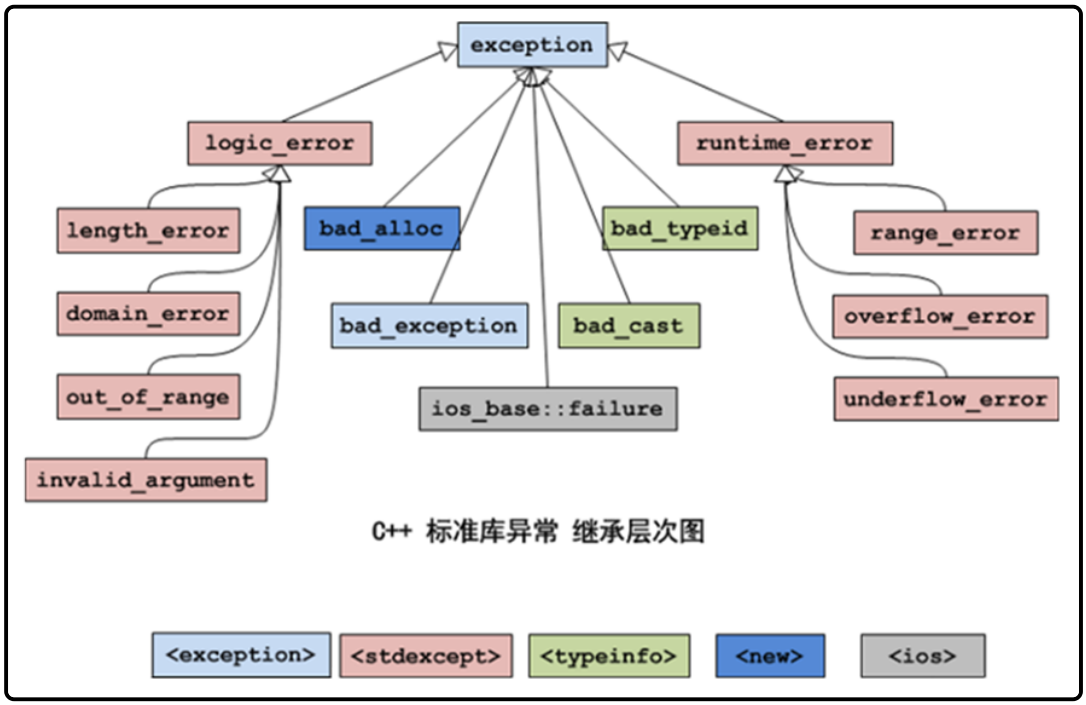
C++ 11 标准弃用了异常规范，使用新增的关键字 **noexcept** 指出函数不会引发异常。

```
void func() noexcept;       //该函数不会抛出异常
```

在实际开发中，大部分程序员懒得再函数后面加 noexcept ，弃用异常已是共识，没必要多此一举。

关键字 **noexcept** 也可以作为运算符，判断表达式（操作符）是否可能引发异常；如果表达式可能引发异常，则返回 false，否则返回 true。

3. C++标准库的异常



new分配内存失败，抛出异常

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

int main(void)
{
    double* ptr=nullptr;

    // 分配一大块内存
    try {
        ptr = new double[10000000000];
        // new分配内存失败，会抛出异常，如果程序中没有处理它抛出的异常，程序会调用 abort() 函数中止
    }

    catch (bad_alloc& e)
    {
        cout << "分配内存失败" << endl;
    }

    if (ptr != nullptr)
    {
        delete[] ptr;
    }
```

```
}
```

实际开发中，用下面这种版本(不抛异常):

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

int main(void)
{
    double* ptr=nullptr;

    ptr = new(std::nothrow) double[10000000000];
    if (ptr == nullptr) cout << "分配内存失败" << endl;

    if (ptr != nullptr)
    {
        delete[] ptr;
    }
}
```

4. 重点关注的异常

4.1 std::bad_alloc

如果内存不足，调用new会产生异常，导致程序中止；如果在 new 关键字后面加 (std::nothrow) 选项，则返回 nullptr，不会产生异常。

4.2 std::bad_cast

dynamic_cast 可以用于引用，但是，没有与空指针对应的引用值，如果转换请求不正确，会出现 std::bad_cast 异常。

4.3 std::bad_typeid

假设表达式 typeid(*ptr)，**当 ptr 是空指针时，如果 ptr 是多态的类型**，将引发 std::bad_typeid 异常。

5. 逻辑错误异常

程序的逻辑错误产生的异常 std::logic_error，通过合理的编程可以避免。

5.1 std::out_of_range

Defines a type of object to be thrown as exception. It reports errors that are consequence of attempt to access elements out of defined range.

It may be thrown by the member functions of std::bitset and s**td::basic_string**, by std::stoi and std::stod families of functions, and by the bounds-checked member access functions (e.g. std::vector::at and std::map::at).

5.2 std::length_error

Defines a type of object to be thrown as exception. It reports errors that result from attempts to exceed implementation defined length limits for some object.

This exception is thrown by member functions of std::basic_string and std::vector::reserve.

5.3 std::domain_error

Defines a type of object to be thrown as exception. It may be used by the implementation to report domain errors, that is, situations where the inputs are outside of the domain on which an operation is defined.

The standard library components do not throw this exception (mathematical functions report domain errors as specified in math_errhandling). Third-party libraries, however, use this. For example, boost.math throws std::domain_error if boost::math::policies::throw_on_error is enabled (the default setting).

5.4 std::invalid_argument

Defines a type of object to be thrown as exception. It reports errors that arise because an argument value has not been accepted.

This exception is thrown by std::bitset::bitset, and the std::stoi and std::stof families of functions.

6. 栈解旋

异常被抛出后，从进入try语句块开始，到异常被抛出之前，这期间在**栈**上构造的所有对象，都会被自动析构。析构的顺序与构造的顺序相反。这一过程称为栈的解旋。

也就是在执行throw前，在try执行期间构造的所有对象被自动析构后，才会进入catch匹配。

在堆上构造的对象，可以用智能指针进行管理动态分配的内存，以便在发生异常时自动释放内存。