


C++ 异常机制分析

5 回复 21 查看




(<https://www.shiyanlou.com/user/8490>) 实验楼管理员  (<https://www.shiyanlou.com/vip>)

4小时前


技术分享 (<https://www.shiyanlou.com/questions/?tag=技术分享>)

文章总结了C++异常机制相关的一些问题，希望对C++学习者有所帮助~

 分享到微博

全部回答



实验楼管理员 (<https://www.shiyanlou.com/user/8490>)  (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

一、C++异常机制概述

异常处理是C++的一项语言机制，用于在程序中处理异常事件。

异常事件在C++中表示为异常对象。异常事件发生时，程序使用 `throw` 关键字抛出异常表达式，抛出点称为异常出现点，由操作系统为程序设置当前异常对象，然后执行程序的当前异常处理代码块，在包含了异常出现点的最内层的 `try` 块，依次匹配 `catch` 语句中的异常对象（只进行类型匹配，`catch` 参数有时在 `catch` 语句中并不会使用到）。

若匹配成功，则执行 `catch` 块内的异常处理语句，然后接着执行 `try...catch...` 块之后的代码。

- 如果在当前的 `try...catch...` 块内找不到匹配该异常对象的 `catch` 语句,则由更外层的 `try...catch...` 块来处理该异常;
- 如果当前函数内所有的 `try...catch...` 块都不能匹配该异常，则递归回退到调用栈的上一层去处理该异常。
- 如果一直退到主函数 `main()` 都不能处理该异常，则调用系统函数 `terminate()` 终止程序。

一个最简单的 `try...catch...` 的例子如下所示。我们有个程序用来记班级学生考试成绩，考试成绩分数的范围在0-100之间，不在此范围内视为数据异常：

```
int main()
{
    int score=0;
    while (cin >> score)
    {
        try
        {
            if (score > 100 || score < 0)
            {
                throw score;
            }

            //将分数写入文件或进行其他操作
        }
        catch (int score)
        {
            cerr << "你输入的分数数值有问题，请重新输入！";
            continue;
        }
    }
}
```

4小时前



二、throw 关键字

在上面这个示例中，throw 是个关键字，与抛出表达式构成了 throw 语句。其语法为：

```
throw 表达式;
```

throw 语句必须包含在 try 块中，也可以是被包含在调用栈的外层函数的 try 块中，如：

```
//示例代码：throw包含在外层函数的try块中

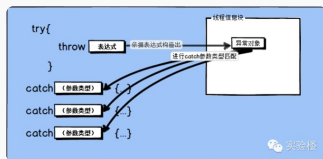
void registerScore(int score)
{
    if (score > 100 || score < 0)
        throw score; //throw语句被包含在外层main的try语句块中
    //将分数写入文件或进行其他操作
}

int main()
{
    int score=0;
    while (cin >> score)
    {
        try
        {
            registerScore(score);
        }
        catch (int score)
        {
            cerr << "你输入的分数数值有问题，请重新输入! ";
            continue;
        }
    }
}
```

执行 throw 语句时，throw 表达式将作为对象被复制构造为一个新的对象，称为**异常对象**。

异常对象放在内存的特殊位置，该位置既不是栈也不是堆，在window上是放在线程信息块TIB中。

这个构造出来的新对象与本级的 try 所对应的 catch 语句进行类型匹配，类型匹配的原则在下面介绍。



在本例中，依据score构造出来的对象类型为int，与 catch(int score) 匹配上，程序控制权转交到 catch 的语句块，进行异常处理代码的执行。如果在本函数内与 catch 语句的类型匹配不成功，则在调用栈的外层函数继续匹配，如此递归执行直到匹配上 catch 语句，或者直到 main 函数都没匹配上而调用系统函数 terminate() 终止程序。

当执行一个 throw 语句时，跟在 throw 语句之后的语句将不再被执行，throw 语句的语法有点类似于 return，因此导致在调用栈上的函数可能提早退出。

4小时前



三、异常对象

异常对象是一种特殊的对象，编译器依据异常抛出表达式复制构造异常对象，这要求抛出异常表达式不能是一个不完全类型（一个类型在声明之后定义之前为一个不完全类型。不完全类型意味着该类型没有完整的数据与操作描述），而且可以进行复制构造，这就要求异常抛出表达式的复制构造函数（或移动构造函数）、析构函数不能是私有的。

异常对象不同于函数的局部对象，局部对象在函数调用结束后就被自动销毁，而异常对象将驻留在所有可能被激活的 `catch` 语句都能访问到的内存空间中，也即上文所说的TIB。当异常对象与 `catch` 语句成功匹配上后，在该 `catch` 语句的结束处被自动析构。

在函数中返回局部变量的引用或指针几乎肯定会造成错误，同样的道理，在 `throw` 语句中抛出局部变量的指针或引用也几乎是错误的行为。如果指针所指向的变量在执行 `catch` 语句时已经被销毁，对指针进行解引用将发生意想不到的后果。

`throw` 出一个表达式时，该表达式的静态编译类型将决定异常对象的类型。所以当 `throw` 出的是基类指针的解引用，而该指针所指向的实际对象是派生类对象，此时将发生派生类对象切割。

除了抛出用户自定义的类型外，C++标准库定义了一组类，用户报告标准库函数遇到的问题。这些标准库异常类只定义了几种运算，包括创建或拷贝异常类型对象，以及为异常类型的对象赋值。

标准异常类	描述	头文件
<code>exception</code>	最基性的异常类，它包含所有其它异常不提供给可操作的消息	<code><exception></code>
<code>runtime_error</code>	异常类运行时才能给出具体的消息	<code><stdexcept></code>
<code>range_error</code>	运行时的错误，产生了超出期望以量化的范围的结果	<code><stdexcept></code>
<code>overflow_error</code>	运行时的错误，计算上溢	<code><stdexcept></code>
<code>underflow_error</code>	运行时的错误，计算下溢	<code><stdexcept></code>
<code>logic_error</code>	程序逻辑错误	<code><stdexcept></code>
<code>domain_error</code>	逻辑错误，参数对已定义的函数是不存在	<code><stdexcept></code>
<code>invalid_argument</code>	逻辑错误，无效参数	<code><stdexcept></code>
<code>length_error</code>	逻辑错误，试图创建一个大超出所期望的大小的对象	<code><stdexcept></code>
<code>out_of_range</code>	逻辑错误，使用一个超出所期望的范围的值	<code><stdexcept></code>
<code>bad_alloc</code>	内存分配失败	<code><new></code>
<code>bad_cast</code>	<code>dynamic_cast</code> 类型转换失败	<code><type_info></code>

四、catch 关键字

`catch` 语句匹配被抛出的异常对象。

- 如果 `catch` 语句的参数是引用类型，则该参数可直接作用于异常对象，即参数的改变也会改变异常对象，而且在 `catch` 中重新抛出异常时会继续传递这种改变。
- 如果 `catch` 参数是传值的，则复制构造函数将依据异常对象来构造 `catch` 参数对象。在该 `catch` 语句结束的时候，先析构 `catch` 参数对象，然后再析构异常对象。

在进行异常对象的匹配时，编译器不会做任何的隐式类型转换或类型提升。除了以下几种情况外，异常对象的类型必须与 `catch` 语句的声明类型完全匹配：

- 允许从非常量到常量的类型转换。
- 允许派生类到基类的类型转换。
- 数组被转换成指向数组（元素）类型的指针。
- 函数被转换成指向函数类型的指针。

寻找 `catch` 语句的过程中，匹配上的未必是类型完全匹配那项，而是在是最靠前的第一个匹配上的 `catch` 语句（我称它为最先匹配原则）。所以，派生类的处理代码 `catch` 语句应该放在基类的处理 `catch` 语句之前，否则先匹配上的总是参数类型为基类的 `catch` 语句，而能够精确匹配的 `catch` 语句却不能够被匹配上。

在 `catch` 块中，如果在当前函数内无法解决异常，可以继续向外层抛出异常，让外层 `catch` 异常处理块接着处理。此时可以使用不带表达式的 `throw` 语句将捕获的异常重新抛出：

```
catch (type x)
{
    //做了一部分处理
    throw;
}
```


被重新抛出的异常对象为保存在TIB中的那个异常对象，与 `catch` 的参数对象没有关系，若 `catch` 参数对象是引用类型，可能在 `catch` 语句内已经对异常对象进行了修改，那么重新抛出的是修改后的异常对象；若 `catch` 参数对象是非引用类型，则重新抛出的异常对象并没有受到修改。

使用 `catch(...){}` 可以捕获所有类型的异常，根据最先匹配原则，`catch(...){}` 应该放在所有 `catch` 语句的最后面，否则无法让其他可以精确匹配的 `catch` 语句得到匹配。

通常在 `catch(...){}` 语句中执行当前可以做的处理，然后再重新抛出异常。注意，`catch`中重新抛出的异常只能被外层的`catch`语句捕获。

4小时前



实验楼管理员 (<https://www.shiyanlou.com/user/8490>)  (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

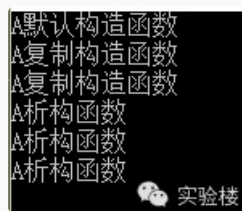
五、栈展开、RAII

其实栈展开已经在前面说过，就是从异常抛出点一路向外层函数寻找匹配的`catch`语句的过程，寻找结束于某个匹配的`catch`语句或标准库函数`terminate`。

这里重点要说的是栈展开过程中对局部变量的销毁问题。我们知道，在函数调用结束时，函数的局部变量会被系统自动销毁，类似的，`throw`可能会导致调用链上的语句块提前退出，此时，语句块中的局部变量将按照构成生成顺序的逆序，依次调用析构函数进行对象的销毁。例如下面这个例子：

```
//一个没有任何意义的类
class A
{
public:
    A() :a(0){ cout << "A默认构造函数" << endl; }
    A(const A& rsh){ cout << "A复制构造函数" << endl; }
    ~A(){ cout << "A析构函数" << endl; }
private:
    int a;
};
int main()
{
    try
    {
        A a ;
        throw a;
    }
    catch (A a)
    {
        ;
    }
    return 0;
}
```

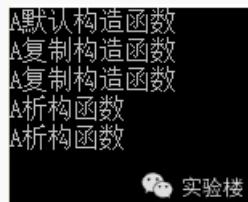
程序将输出：



定义变量`a`时调用了默认构造函数，使用`a`初始化异常变量时调用了复制构造函数，使用异常变量复制构造`catch`参数对象时同样调用了复制构造函数。三个构造对应三个析构，也即`try`语句块中局部变量`a`自动被析构了。然而，如果`a`是在自由存储区上分配的内存时：

```
int main()
{
    try
    {
        A * a= new A;
        throw *a;
    }
    catch (A a)
    {
        ;
    }
    getchar();
    return 0;
}
```

程序运行结果：



同样的三次构造，却只调用了两次的析构函数！说明a的内存在发生异常时并没有被释放掉，发生了内存泄漏。RAII机制有助于解决这个问题，RAII（Resource acquisition is initialization，资源获取即初始化）。它的思想是以对象管理资源。为了更方便、鲁棒地释放已获取的资源，避免资源死锁，一个办法是把资源数据用对象封装起来。程序发生异常，执行栈展开时，封装了资源的对象会被自动调用其析构函数以释放资源。C++中的智能指针便符合RAII。关于这个问题详细可以看《Effective C++》条款13。

六、异常机制与构造函数

异常机制的一个合理的使用是在构造函数中。构造函数没有返回值，所以应该使用异常机制来报告发生的问题。更重要的是，构造函数抛出异常表明构造函数还没有执行完，其对应的析构函数不会自动被调用，因此析构函数应该先析构所有所有已初始化的基对象，成员对象，再抛出异常。

C++类构造函数初始化列表的异常机制，称为function-try block。一般形式为：

```
myClass::myClass(type1 pal)
    try: _myClass_val (初始化值)
{
    /*构造函数的函数体 */
}
    catch ( exception& err )
{ /* 构造函数的异常处理部分 */
};
```

4小时前



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) 💎 (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

七、异常机制与析构函数

C++不禁止析构函数向外界抛出异常，但析构函数被期望不向外界函数抛出异常。析构函数中向函数外抛出异常，将直接调用 `terminator()` 系统函数终止程序。

如果一个析构函数内部抛出了异常，就应该在析构函数的内部捕获并处理该异常，不能让异常被抛出析构函数之外。可以如此处理：

- 若析构函数抛出异常，调用 `std::abort()` 来终止程序。
- 在析构函数中catch捕获异常并作处理。

关于具体细节，有兴趣可以看《Effective C++》条款08：别让异常逃离析构函数。

八、noexcept 修饰符与 noexcept 操作符

`noexcept`修饰符是C++11新提供的异常说明符，用于声明一个函数不会抛出异常。

编译器能够针对不抛出异常的函数进行优化，另一个显而易见的好处是你明确了某个函数不会抛出异常，别人调用你的函数时就知道不用针对这个函数进行异常捕获。

在C++98中关于异常处理的程序中你可能会看到这样的代码：

```
void func() throw(int ,double ) {...}
void func() throw(){...}
```

这是throw作为函数异常说明，前者表示 `func()` 这个函数可能会抛出int或double类型的异常，后者表示 `func()` 函数不会抛出异常。

事实上前者很少被使用，在C++11这种做法已经被摒弃，而后者则被C++11的noexcept异常声明所代替：

```
void func() noexcept{...}  
//等价于  
void func() throw(){...}
```

在C++11中，编译器并不会在编译期检查函数的noexcept声明，因此，被声明为noexcept的函数若携带异常抛出语句还是可以通过编译的。

在函数运行时若抛出了异常，编译器可以选择直接调用 `terminate()` 函数来终结程序的运行，因此，noexcept的一个作用是阻止异常的传播,提高安全性.

上面一点提到了，我们不能让异常逃出析构函数，因为那将导致程序的不明确行为或直接终止程序。实际上出于安全的考虑，C++11标准中让类的析构函数默认也是noexcept的。同样是为了安全性的考虑，经常被析构函数用于释放资源的delete函数，C++11也默认将其设置为noexcept。

noexcept也可以接受一个常量表达式作为参数，例如：

```
void func() noexcept(常量表达式);
```

常量表达式的结果会被转换成bool类型， `noexcept(bool)` 表示函数不会抛出异常， `noexcept(false)` 则表示函数有可能会抛出异常。故若你想更改析构函数默认的noexcept声明，可以显式地加上 `noexcept(false)` 声明，但这并不会带给你什么好处。

九、异常处理的性能分析

异常处理机制的主要环节是运行期类型检查。当抛出一个异常时，必须确定异常是不是从try块中抛出。

异常处理机制为了完善异常和它的处理器之间的匹配，需要存储每个异常对象的类型信息以及catch语句的额外信息。

由于异常对象可以是任何类型（如用户自定义类型），并且也可以是多态的，获取其动态类型必须要使用运行时类型检查（RTTI），此外还需要运行期代码信息和关于每个函数的结构。

当异常抛出点所在函数无法解决异常时，异常对象沿着调用链被传递出去，程序的控制权也发生了转移。

转移的过程中为了将异常对象的信息携带到程序执行处（如对异常对象的复制构造或者catch参数的析构），在时间和空间上都要付出一定的代价，本身也有不安全性，特别是异常对象是个复杂的类的时候。

异常处理技术在不同平台以及编译器下的实现方式都不同，但都会给程序增加额外的负担，当异常处理被关闭时，额外的数据结构、查找表、一些附加的代码都不会被生成，正是因为如此，对于明确不抛出异常的函数，我们需要使用noexcept进行声明。

感谢您的耐心阅读。

文章地址：<http://www.linuxidc.com/Linux/2016-01/127589.htm>

作者：QG-whz

4小时前

登录后才能回答问题哟~

我要提问

标签

- Linux (<https://www.shiyanlou.com/questions/?tag=Linux>) Python (<https://www.shiyanlou.com/questions/?tag=Python>)
- C/C++ (<https://www.shiyanlou.com/questions/?tag=C/C++>) 实验环境 (<https://www.shiyanlou.com/questions/?tag=实验环境>)
- 技术分享 (<https://www.shiyanlou.com/questions/?tag=技术分享>) 功能建议 (<https://www.shiyanlou.com/questions/?tag=功能建议>)
- 课程需求 (<https://www.shiyanlou.com/questions/?tag=课程需求>) Java (<https://www.shiyanlou.com/questions/?tag=Java>)
- 其他 (<https://www.shiyanlou.com/questions/?tag=其他>) SQL (<https://www.shiyanlou.com/questions/?tag=SQL>)
- NodeJS (<https://www.shiyanlou.com/questions/?tag=NodeJS>) Hadoop (<https://www.shiyanlou.com/questions/?tag=Hadoop>)
- 常见问题 (<https://www.shiyanlou.com/questions/?tag=常见问题>) Web (<https://www.shiyanlou.com/questions/?tag=Web>)
- Shell (<https://www.shiyanlou.com/questions/?tag=Shell>) PHP (<https://www.shiyanlou.com/questions/?tag=PHP>)
- Git (<https://www.shiyanlou.com/questions/?tag=Git>) HTML (<https://www.shiyanlou.com/questions/?tag=HTML>)
- HTML5 (<https://www.shiyanlou.com/questions/?tag=HTML5>) 信息安全 (<https://www.shiyanlou.com/questions/?tag=信息安全>)
- 网络 (<https://www.shiyanlou.com/questions/?tag=网络>) GO (<https://www.shiyanlou.com/questions/?tag=GO>)
- NoSQL (<https://www.shiyanlou.com/questions/?tag=NoSQL>) Android (<https://www.shiyanlou.com/questions/?tag=Android>)
- 训练营 (<https://www.shiyanlou.com/questions/?tag=训练营>) Ruby (<https://www.shiyanlou.com/questions/?tag=Ruby>)
- Perl (<https://www.shiyanlou.com/questions/?tag=Perl>)

相关问题

- 编译器的工作过程 (<https://www.shiyanlou.com/questions/3089>)
- 你需要知道的12个Git高级命令 (<https://www.shiyanlou.com/questions/3088>)
- [译]学习Python编程的19个资源 (<https://www.shiyanlou.com/questions/3084>)
- 7 款超具个性的 HTML5 播放器 (<https://www.shiyanlou.com/questions/3082>)
- Java程序员使用的20几个大数据工具之数据库 (<https://www.shiyanlou.com/questions/3068>)

动手做实验，轻松学IT。

实验楼-通过动手实践的方式学会IT技术。

公司简介 (<https://www.shiyanlou.com/aboutus>) 联系我们 (<https://www.shiyanlou.com/contact>) 常见问题 (<https://www.shiyanlou.com/faq#howtostart>)
我要开课 (<https://www.shiyanlou.com/labs>) 隐私协议 (<https://www.shiyanlou.com/privacy>) 会员条款 (<https://www.shiyanlou.com/terms>)
友情链接 (<https://www.shiyanlou.com/friends>)
站长统计 (http://www.cnzz.com/stat/website.php?web_id=5902315) 蜀ICP备13019762号 (<http://www.miibeian.gov.cn/>)



QQ群



微信



微博

(<http://weibo.com/shiyanlou2013>)