C++ 异常处理(三) exception类_vc exception 没定义_运妙心藏的博客-CSDN博客

成就一亿技术人!

1,exception

头文件:

#include <exception>

C++ 可以把它用作其它异常类的基类。

代码可以引发exception异常，也可以把exception用作基类，

在从exception派生而来的类中重新定义一个名为what()的虚拟成员函数，

它返回一个字符串，该字符串随实现而异。

```
#include <exception>
class bad_hmean : public std::exception
{
public:
const char* what()
        {
return "bad arguments to hmean()";
        }

};
class bad_gmean : public std::exception
{
public:
const char* what()
        {
return "bad arguments to gmean()";
        }

};
```

如果不想以不同的方式捕获这些派生来的异常，可以在同一个基类处理程序中捕获它们:

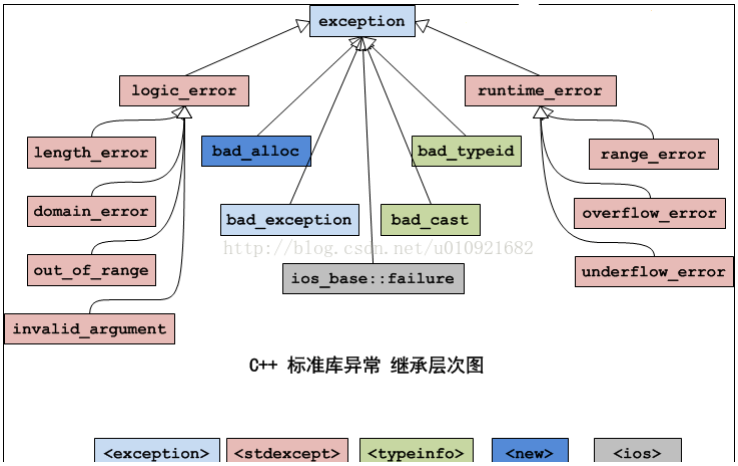
```
try{

}
catch(std::exception& e)
{
        cout << e.what() << endl;

}
```

否则，应分别捕获它们。

C++库定义了很多基于exception的异常类型。

1，stdexcept 异常类

该文件定义了logic_error和runtime_error类，它们都是以公有类从exception派生而来的。



每个类所在的**头文件**在图下方标识出来.

标准异常类的成员：

① 在上述继承体系中，每个类都有提供了构造函数、复制构造函数、和赋值操作符重载。

② logic_error类及其子类、runtime_error类及其子类，它们的构造函数是接受一个string类型的形式参数，用于异常信息的描述；

③ 所有的异常类都有一个what()方法，返回const char* 类型（C风格字符串）的值，描述异常信息。

标准异常类的具体描述：

异常名称	**描述**
exception	所有标准异常类的父类
bad_alloc	当operator new and operator new[]，请求分配内存失败时
bad_exception	这是个特殊的异常，如果函数的异常抛出列表里**声明了bad_exception异常**，当函数内部抛出了异常抛出列表中**没有的**异常，这是调用的unexpected函数中若抛出异常，不论什么类型，都会**被替换为bad_exception类型**
bad_typeid	使用typeid操作符，操作一个NULL指针，而该指针是带有虚函数的类，这时抛出bad_typeid异常
bad_cast	使用dynamic_cast转换**引用**失败的时候
ios_base::failure	io操作过程出现错误
logic_error	逻辑错误，可以在**运行前**检测的错误
runtime_error	运行时错误，仅在**运行时**才可以检测的错误

logic_error的子类：

异常名称	**描述**
length_error	试图生成一个超出该类型最大长度的对象时，例如vector的resize操作
domain_error	参数的值域错误，主要用在数学函数中。例如使用一个负值调用只能操作非负数的函数
out_of_range	超出有效范围
invalid_argument	参数不合适。在标准库中，当利用string对象构造bitset时，而string中的字符不是'0'或'1'的时候，抛出该异常

runtime_error的子类：

异常名称	**描述**
range_error	计算结果超出了有意义的值域范围
overflow_error	算术计算上溢
underflow_error	算术计算下溢

例：logic_error下的

domain_error：定义域由参数的可能取值组成

invalid_argument：值域由函数可能的返回值组成，指函数传递了一个意料之外的值

length_error：指没有足够的空间来执行所需的操作，如string类的append()方法合并得到的字符串长度超过最大允许长度会引发该异常

out_of_bounds：通常指索引错误，如定义一个数组，其operator[]在使用索引无效时会引发该异常

一般而言：

logic_error系列异常表明存在可以通过编程修复的问题。

runtime_error系列异常表明存在无法避免的问题。

2,对于bad_alloc 异常与new

对于使用new导致的内存分配问题，C++的最新处理方式是让new引发bad_alloc异常。

头文件new包含bad_alloc类的声明，它从exception公有派生来。

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

struct Big
{
double stuff[20000];
};

int main()
{
        Big* pb;
try
        {
                cout << "Trying to get a big block of memory:\n";
                pb = new Big[11000];
        }
catch(bad_alloc& ba)
        {
                cout << "Caught the exception!\n";
```

```
                cout << ba.what() << endl;
exit(EXIT_FAILURE);
        }
        cout << "Memory successfully allocated\n";
        pb[0].stuff[0] = 4;
        cout << pb[0].stuff[0] << endl;
delete [] pb;
return 0;
}
```

输出：

```
Trying to get a big block of memory:
Caught the exception!
bad allocation
```

这里，方法what()返回的字符串“std::bad_alloc".

3, 空指针与new

例：int* pi = new (std::nothrow) int;

int* pa = new (std::nothrow) int[500];

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

struct Big
{
double stuff[20000];
};

int main()
{
        Big* pb;
        pb = new (std::nothrow) Big[11000];
if (pb == 0)
        {
                cout << "Could not allocate memory.Bye!\n";
exit(EXIT_FAILURE);
        }
        cout << "Memory successfully allocated\n";
        pb[0].stuff[0] = 4;
        cout << pb[0].stuff[0] << endl;
delete [] pb;
return 0;
}
```

输出：

```
Could not allocate memory.Bye!
```