Basics on Exception

异常是程序中经常会遇到的一些情况：

- 除数为0
- 数组越界
- 内存分配异常

处理指的是，可以给出错误提示信息

- 然后让程序沿着一条不会出错的路径继续执行；
- 或者直接终止程序，结束前可能做一些工作：
    - 将内存中的数据写入文件
    - 关闭打开的文件
    - 释放动态分配的内存空间等

通常，异常会分散在各处（尤其是对于大型程序而言）进行异常处理不利于代码维护，尤其是不同地方发成同一种异常，都要编写相同的处理代码也是一种不必要的重复和冗余。因而，C++引入异常处理机制。

C++中，异常处理基本思想：函数A在执行过程中发现异常可以不加处理，而是抛出（throw）一个异常给A的调用者，假定为函数B。

抛出异常后，不加处理会导致A立即终止，在这种情况下，函数B可以选择捕获A抛出的异常进行处理，也可以选择置之不理。此时这个异常会抛给B的调用者。依次类推。

最终，异常可能会一层层传递给最外层的main函数。main函数应当处理异常，如不处理，则程序立即异常终止。

异常处理

异常处理机制（exception handling）允许程序中独立开发的部分能够在运行时就出现的问题进行通信并作出相应的处理。异常使得我们能够将问题的检测与解决过程中分离开来。程序的一部分负责检测问题的出现，然后解决问题的任务传递给程序的另一部分。检测环节不需要知道问题处理模块的细节，反之亦然。

抛出异常

C++通过抛出**一条表达式**来引发（raise）一个异常。被抛出的表达式的类型和当前的调用链共同决定哪段**处理代码（handler）将被用来处理该异常。被选中的handler是在调用链中与抛出对象类型匹配的最近的handler。

当执行一个throw时，throw后面的语句不再执行。程序的控制权从throw转到与之匹配的catch模块。该catch可能是同一个函数中的局部catch，也可能是位于直接或间接调用发生该异常的函数的另一个函数中的catch。控制权从一方转移到另一处有两个重要含义：

- 沿着调用链的函数可能会提早推出。
- 程序一旦终止，执行handler，则沿着调用链创建的对象将被销毁。

因为throw后面的语句不再执行，有点类似与return语句。它通常作为条件语句的一部分或者作为某个函数的最后（唯一）一条语句。

栈展开

当throw一个异常之后，程序暂停当前函数的执行过程并立即开始寻找与异常匹配的catch子句。当throw出现在一个try语句块内时，检查与该try块关联的catch子句。如果找到了匹配的catch，则使用该catch处理异常。如果没有找到，且该try语句嵌套在其他try块中，则继续检查与外层try匹配的catch子句。还是未找到，则退出当前函数，调用该函数的调用者的外层函数中继续寻找。

此过程称之为“栈展开”。此时，该过程中的局部对象（包括类对象）将会被销毁。

应当注意的是，异常发生在构造函数中时，此时，当前对象可能只构造了一部分。有的成员已经初始化，有的还未初始化。这时，编译器应当确保已经构造的成员可以正确的销毁。

析构函数与异常

析构函数总是会被执行（因为要回收资源），但是函数中负责释放资源的代码有可能被跳过。例如，如果一个块分配了资源，并且在负责释放这些资源的代码前面发生了异常，则释放资源的代码将不会被执行。另一方面，类对象分配的资源将由类的析构函数负责释放。因此，如果我们使用类来控制资源的分配，就能确保无论函数正常结束还是遭遇异常，资源都可被正确的释放。

析构函数在栈展开的过程中执行，这一事实影响着我们编写析构函数的方式。在栈展开的过程中，已经引发了异常，但是没有处理它。如果异常抛出后没有被正确地捕获，则系统调用terminate函数，终止程序。因此，**出于栈展开可能使用析构函数的考虑，析构函数不应该抛出不能被它自身处理的异常**。也就是说，如果析构函数需要执行某个可能抛出异常的操作，则该操作应该被放置在一个try块中，并且在该析构函数内部得到处理。

❗ 实际中，析构函数仅仅负责释放资源，所以不太可能抛出异常。所有标准库类型都能确保它们的析构函数不会引发异常。

> 栈展开的过程中，运行类类型的局部对象的析构函数，因为这些析构函数是自动执行的，所以它们不应该抛出异常。一旦在栈展开的过程中析构函数抛出异常，并且析构函数自身没能捕获该异常，则程序将被终止（资源将会被更底层的操作系统负责回收）。

这里先给出try和catch对异常处理的代码：

```
#include <iostream>
using namespace std;
int main(){
    double m,n;
    cin>>m>>n;
    try{
        if(n==0)
            throw -1;
        else cout<<m/n;
    }
    catch (double d){
        ...
    }
```

```
    catch (int e){
        ...
    }
}
```

- 当n为0的时候，抛出异常，该整型异常会被类型匹配的第一个catch捕获，即进入catch(int e)块执行。

如果catch(int e)改为catch(char e)，则n为0的时候，抛出的整型异常就没有catch块捕获，这个异常就得不到处理，即main函数中无catch块捕获该异常，则终止程序。

异常对象

异常对象 (exception object) 是一种特殊对象，编译器使用异常抛出表达式来对异常对象进行拷贝初始化。因此，throw的语句中的表达式必须拥有完全类型。

- 如果该表达式是类类型的话，则相应的类必须含有一个可访问的析构函数和一个可访问的拷贝或移动构造函数。(析构异常对象需要访问那个类对象的析构函数，用那个类对象拷贝初始化异常对象需要访问类的拷贝或移动构造函数)
- 如果该表达式是数组类型或函数类型，则表达式将转换成与之对应的指针类型。

不需要关注该对象是怎么样的，只需要知道该对象很特殊即可，并且使用抛出的表达式对象进行拷贝初始化。

异常对象位于由编译器管理的空间中，编译器确保无论最终调用的是哪个catch子句都能访问该空间，当异常处理完毕后，异常对象被销毁。

如前所述，异常被抛出时，沿着调用链的块将依次退出直至找到与异常匹配的handler。如果退出了某个块 (如try块) ，则同时释放块中局部对象使用的内存。因此，抛出一个指向局部对象 (如try块中的局部对象) 的指针，几乎肯定是一种错误的行为。这是因为，如果指针所指的对象位于某个块中，而该块在catch语句之前就已经退出了，则意味着在执行catch之前局部对象已经被销毁了。

18.1.1 练习：为了理解异常对象

```
a) range_error r("error"); throw r;
// 此时，r被用来初始化异常对象，此时异常对象是range_error类型
b) exception *p=&r; throw *p;
// 此时，*p被用来初始化异常对象，而*p是exception类型，故。。。
c) ... throw p;
// 此时，异常对象是exception *类型
```

捕获对象

catch子句 (catch clause) 中的异常声明 (exception)

```
catch(int e){
    //...
}
```

看起来像是只包含了一个形参的函数形参列表，当然，如果catch块中无须访问抛出的表达式，则我们可以忽略捕获形参的名字。

声明的类型决定了处理的代码所能捕获的异常类型。这个类型必须是

- 完全类型
- 可以是左值引用，但不可以是右值引用

进入一个catch语句，通过异常对象初始化异常声明中的捕获形参。如果，参数是引用类型，此时改变参数也会改变异常对象。

❗ 如果catch的参数是基类类型，则我们可以使用其派生类类型的异常对象对其进行初始化。此时，如果catch的参数是非引用类型，则异常对象将被切掉一部分。这与派生类对象以值传递的方式给一个普通函数差不多。此外，如果catch的参数是基类的引用，则该参数将以常规方式绑定到异常对象上。

❗ 异常声明的静态类型决定catch语句所能执行的操作。如果catch的参数是基类类型，则catch无法使用派生类特有的任何成员。

查找匹配的handler