1. 异常的基本思想：

让一个函数在发现了自己无法处理的错误时抛出（throw）一个异常，然后它的（直接或者间接）调用者能够处理这个问题。

将问题检测和问题处理相分离。 在异常处理过程中，由问题检测代码可以抛出一个对象给问题处理代码，通过这个对象的类型和内容，实际上完成了两个部分的通信，通信的内容是“出现了什么错误”。

1. 异常的注意事项

抛出异常，，必须要处理。如果不处理，就会中断程序。所以异常使得开发出来的程序更加健壮。

* 如果匹配的处理未找到，则运行函数terminate将自动被调用，其缺省功能调用abort终止程序。
* 如有多个异常对象，用多个对应的catch语句进行捕获。Catch语句会根据异常对象出现的先后顺序进行检查。匹配的catch语句捕获并处理异常(或继续抛出异常)
* 处理不了的异常，可以在catch的最后一个分支，使用throw，向上抛。

异常处理时可以跳级处理，如果不想处理，只写throw，让上一层处理。总之异常最终一定要在某一层进行处理，不然，程序就会终止。

Catch(…)省略号代表所有其他类型

1. 处理异常语法：

在try代码块中进行正常逻辑处理，并且在会出错的位置抛出一个异常对象，在try代码块之下catch捕获响应的异常对象，输出错误的原因。这样就避免了程序的终止，也便于后续的修改。

Int myDevide(int a, int b)

{  
 if( b == 0)

Throw -1; //抛出异常

Return a / b;

}

try

{

myDevide(a, b);

}

Catch(int)

{  
 cout << “int类型异常” << endl;

}

1. 自定义异常类

自己定义一个异常类，那么就可以将某一类异常整合到一个类中。

示例如下：

Class MyException

{  
public:

Void printError()

{  
 cout << “自定义异常” << endl;;

}

}

Int myDevide(int a, int b)

{  
 if( b == 0)

{

Throw myException(); //匿名对象

}

Return a / b;

}

Catch(MyException e)

{  
 e.printError();

}

1. 栈解旋

所谓栈解旋，从进入try代码块起，到异常对象被抛出前，在这期间的创建的栈对象都会被自动地析构。析构的顺序与构造的顺序相反。

如下所示：

Try

{

Person p1;

Person p2;

Throw myException();

}

P1，p2都是在栈上开辟的对象，因此在抛出myException()异常对象前，这两个栈对象就会被自动析构。

六．异常接口声明

1.当我们在声明函数时，可以直接在参数框后面列出该函数可能抛出的异常类型：  
void func() throw(A, B) {} //这个函数func只能抛出A，B类型异常及其子类型。

2.如果函数声明时没有包含异常接口声明，则此函数以抛任何类型的异常:

如：void func() {}

3.如果函数声明包含了异常接口声明，但是没有写任何异常类型，说明该函数不能抛出任何类型异常：

Void func() throw() {}

4.如果函数抛出了它的异常接口没有声明的异常类型，那么unexpected函数会被调用，该函数默认调用terminate函数中断程序。

七．异常变量的生命周期

当我们抛出异常变量时，可能会抛出变量的引用，指针，对于每一种不同类型的异常变量，它的声明周期都是不同的。

如：

1. throw( MyException() )，catch(MyException e)，抛出异常时调用默认构造，那么捕获异常时，就会多调用一次拷贝构造函数和之后的析构函数。且默认构造和拷贝构造的析构都是在捕获异常之后。
2. throw( &MyException() ), catch(MyException \*e)，抛出异常对象的指针，捕获异常对象的指针，因为该异常对象开辟在了栈空间上，那么该异常对象在捕获异常前就会被释放。
3. throw( new MyException() )，catch(MyException \*e)，用new在堆区上开辟对象，抛出指针，则该异常对象不会被自动释放，而是由程序员自己释放。
4. throw( MyException() ), catch(MyException &e)，捕获异常对象的引用，那么只需要抛出异常的默认构造，就不会调用拷贝构造了，提高速度。推荐使用引用的方式来捕获异常。

八. 异常的多态使用

多态的思想我们之前已经介绍过了，便于功能的拓展。当异常与多态结合起来时，就可以将不同的异常整合成不同的子类，并继承于一个公共的基类，便于之后的拓展。通过一个接口printError可以调用不同的异常提示，抛出不同的错误对象。

如下所示：

Class BaseException

{  
 public:

Virtual void printError(){}

}

Class NullPointerException : public BaseException

{  
public:

Virtual void printError()

{

Cout << “空指针异常” << endl;

}

}

Void test()

{

Try

{

Throw NullPointerException(); //匿名对象

}

Catch(BaseException & e) //基类的指针或引用指向子类

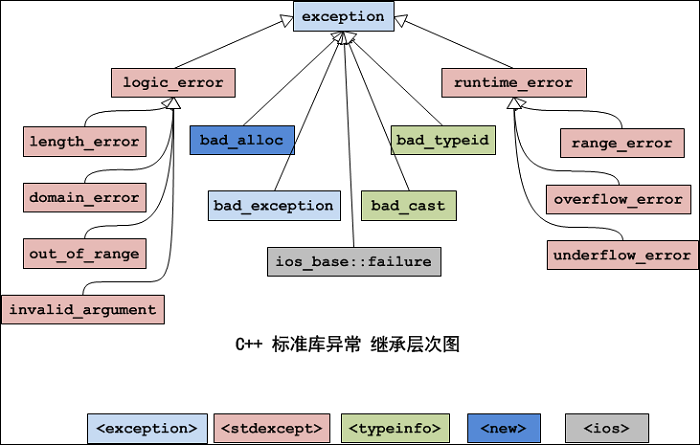
{  
 e.printError(); //调用子类的函数

}

}

九．系统标准异常

如何使用系统自带的异常？系统标准异常也是从exception异常基类开始继承的，异常类继承层级结构图如下：



1. 在上述继承体系中，每个类都有提供了构造函数、拷贝构造函数、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。

注意：要调用系统标准异常，就要#include<stdexcept>，其他的与自定义的异常使用方法一样。