异常机制

1. **调用 abort () ：**

作用 ： 直接终止程序 ；

1. **异常机制 :**

引发异常 ： throw ;

捕获异常 ： catch ;

Try 块 : try ;

1. **栈解退 ：**

假设函数由于出现异常而终止 ， 则程序将释放栈中的内存 ， 但不会在释放栈的第一个返回地址后停止 ， 而是继续释放栈 ， 直到找到第一个 try 块 中的返回地址 ， 然后， 控制权转让给了异常处理程序 ， 而不是调用后面的第一条语句 ；

当抛出了异常，但还没在特定的作用域中被捕获时，函数调用堆栈便被“解退”，并试图在下一个外层try...catch代码中捕获这个异常。解退函数调用堆栈意味着抛出未捕获异常的那个函数将终止，这个函数中的所有局部变量都将销毁，控制会返回到原先调用这个函数的语句。

如果有一个try代码块包含了这条语句，则它就会试图捕获这个异常。如果没有代码块包含这条语句，则堆栈解退再次发生。如果没有任何catch处理器捕获这个异常，则会调用terminate函数，终止程序。

**注意： 栈解退后此栈若被释放 ， 所有变量将不存在 ， 调用引用的话将引用此副本 ；**

1. **exception 类 :**

**exception 头文件 中定义了 exception 类 ；**exception 类中定义了一个 what () 虚函数 ， 他返回一个字符串 ；该实现虽现实而已

1. **stdexcept 异常类 ；**

头文件 stdexcept 定义了其他的几个异常类 ， 首先 ， 该文件中定义了 logic\_error 类和runtime\_error 类; 他们都是以共有方式从exception 中派生来的 ；

**logic\_error :**

**domain\_error :**

如果函数不再定义域之内 ， 则引发 domain\_error 异常 ；

**invalid\_arguement :**

如果给函数传递一个意料外的值 ， 则引发 invalid\_arguement 异常 ；

**length\_error :**

如果没有足够的空间完成所需的操作则引发 length\_error 异常；

**out\_of\_bounds :**

通常用于当引索错误时 ， 引发out\_of\_bounds 异常 ；

**runtime\_error:**

**runge\_error :**

非出现上溢或下溢情况的浮点错误 ；

**overflow\_error :**

出现上溢错误 ；

**underflow\_error :**

出现下溢错误 ;

1. **bad\_alloc 异常 ：**

头文件 new中包含了 bad\_alloc 异常他是从 exception 类中共有派生而来的 ；

作用 ： 当分派内存失败时 则会引发 bad\_alloc 异常 ；

1. **异常与迷失方向:**
2. **未捕获异常 ：**

如果类型不是在函数中引发的 （或者函数没有异常规范），则必须捕获他 ，如果没有捕获（没有 try 或 对应的 catch 块时 ， 将出现这种情况）

未捕获异常不会导致程序立刻终止 ， 相反程序首先调用函数 terminate() ;默认情况下 ， terminate 调用 abort () 函数终止程序进程 ；

可以调用 set\_terminate ()函数来制定terminate () 应调用的函数 ；

**注意 ： terminate 跟 set\_terminate 都是位于头文件exception**

使用方法 ： set\_terminate() 将没有返回值不带任何参数的函数作为参数， 如果设置多个

terminate() 将调用最后一次设置 ；

如： Myquit （）｛。。。。。。。。。。。。。。。｝；

ste\_terminate(Myquit) ;

1. **意外异常 ：**

如果它是在异常规范的函数中引发的 ，则必须与规范列表中的某种异常相匹配 ， 否则将出现意外异常 ；

**RTTI :**

**dynamic\_cast 运算符 ：**

例如 ：

Superb \* pm = dynamic\_case<Superb \*>(pm) ;

指针 pg 能否安全转换为 Superb 指针 如果可以则返回对象地址 否则 返回空指针 ；

**typeid 运算符 和 type\_info 类 ：**

typeid (Magnificent) == typeid (\* pd) // 比较两对象是否为同一对象 ；（ture/fulse）

**注意：如果 pd 为空指针将引发 bad\_typeid 异常 ； （头文件 typeinfo）**

**类型转换运算符 ：**

**dynamic\_cast :**

dynamic\_cast <type-name>(expression) ;

功能 ： 允许类层次结构进行向上转换， 而不允许其他转换 ；

**const\_cast:**

const\_cast <type-name> (experssion) ;

功能： 可以修改 const.volatile

注意 ： 如果是一个const 指针指向 const 常量 则修改后 无法改变值 ；

**static\_cast :**

static\_cast <type-name>(experssion)

仅当 type-name 可以隐式转换为 experssion 或experssion 可以隐式转换为 type-name时方可

转换

**reinterpert\_cast:**

见书中详解 ；