|  |  |
| --- | --- |
| 1. 抛出异常 2. 栈展开 3. 栈展开过程中对象被自动销毁 4. 析构函数与异常 5. 异常对象 | 抛出一条表达式引发一个异常  处理代码  Trycatch语句块  栈展开  Terminate  异常对象  Throw |
| 1.抛出一条表达式来引发一个异常。被抛出的表达式的类型以及当前调用链共同决定了哪段处理代码来处理异常。  被选中的处理代码是在调用链中与抛出对象类型匹配最近的处理代码。  2.Throw后面的语句不在被执行。程序的控制权从throw转移到与之匹配的catch模块。也许在当前，也许在调用这个函数的另一个函数中。  3.控制权从一处转到另一处有两个重要意义：  1）沿着调用链的函数可能会提早退出  2）一旦程序开始执行异常处理代码，则沿着调用链创建的对象将被销毁   1. 栈展开 2. 抛出异常后，程序暂停当前函数的执行过程并开始寻找与之异常匹配的catch子句，若没找到则继续检查外层try匹配的catch，退出当前的函数，在调用当前函数的外层函数中继续寻找，以此类推，称为栈展开 3. 退出主函数后查找过程终止 4. 若找到了catch执行catch中的代码，找到与try块关联的最后一个catch子句之后的点，从这里继续执行，就是catch所在的try外 5. 没找到，程序退出，异常被认为妨碍程序正常执行的事件，则调用terminate,终止当前的程序 6. 栈展开过程中对象被自动销毁 7. 位于链上的语句块可能会提前退出，编译器负责确保在这个块中创建的对象能被正确的销毁，若局部对象的类型是类类型，则析构函数将被自动调用 8. 若异常在构造函数中，得确保构造了一部分的成员能被析构销毁 9. 异常也可能在数组或容器初始化过程中，也得确定构造的一部分成员能被正确销毁 10. 析构函数与异常 11. 析构函数总是会被执行，所以可以在析构函数中进行对类的资源销毁是好的写法，能确保函数正常结束还是遭遇异常都能正确释放资源，不然在函数中分配资源，而在销毁资源之前抛出异常就不会销毁资源了。 12. 栈展开过程中析构函数可能会执行。因为析构函数在销毁局部对象时自动执行，但如果此时执行析构函数，析构函数中抛出异常并且没有处理，则会直接造成terminate异常，此时的话就退出了。所以我们规定：析构函数不该抛出异常。并且若要抛出异常，一定要在析构函数中进行良好处理，不然程序立刻停止。   所有标准库类型都能确保他们的析构函数不会引发异常。   1. 异常对象 2. 是一种特殊的对象，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 3. Throw表达式必须具有完全类型。 4. 若该表达式是类类型，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。若是数组或函数，会转换为指针 5. Throw不能抛出局部对象，并且不能返回指向局部对象的指针 6. 静态编译时的类型决定了异常对象的类型。若抛出解引用一个基类指针，虽然动态类型是派生类，但也只有基类部分抛出（若抛出基类解引用表达式）派生类被切掉   重要思想：   1. throw抛出表达式需要具有完全类型，就是有定义有声明的类或者函数或者\*p要有分配的资源 2. 在发生异常，必须确定能销毁已经构造的成员，但是动态分配的资源不会自动删除，需要放置在类中处理，或者用动态指针 | |