**C++学习笔记——异常处理**

**1. try-throw-catch**

C++使用try-throw-catch组合来捕获异常。

图形用户界面, 文本, 应用程序

描述已自动生成

在try块中，如果引发某个异常，类型与该异常的类型匹配的第一个关联catch块将捕获该异常。换言之，执行将从throw语句跳转到catch语句。

如果未找到可用的catch块，则调用std::terminate并且程序会退出。

最后，C++既不提供也不需要finally块来确保在引发异常时释放所有资源。合理的C++代码，应当使用对象来管理资源，并在对象的析构函数中调用释放资源的代码，从而确保出现异常时，资源一定会被释放。

**2. assert**

断言。来自于C语言。使用时需要包含头文件<cassert>。

assert本身含义是：断言某个表达式永远为true。实际应用时，多用于检查开发过程中永远不应为true的条件。例如：除数为零。使用异常来处理此类错误是没有意义的，因为错误指示的是代码中必须修复的问题。它并不表示程序在运行时必须从中恢复的状态。

assert在运行时生效。

**3. static\_assert**

C++11引入的关键字，功能类似于assert，但仅适用于常量表达式，在编译而非运行期间就可以检查断言。如果违反static\_assert规则，编译将不能通过。

常与constexpr或模板搭配使用。

**4. noexception**

C++11引入的关键字，通常用于函数声明中，表示该函数执行过程中不可能抛出异常。其作用，一是提升代码可读性；二是当编译器遇到noexception函数时，可能会基于“该函数不会抛出异常”的假设，进行相应优化。

如果函数可能抛出异常，应当使用noexception(false)。

这里以C++的标准operator new和operator delete声明为例：

void\* operator new(std::size\_t size) noexcept(false); // new执行时可能抛出异常

void operator delete(void\* pMem) noexcept; // 保证delete不会抛出异常