C++期末不挂科

CH7 - 异常处理

本章内容

- 异常 vs Bug
- throw 和 try - catch
- 自定义异常类

理解以下名词：

- 异常
- 异常的抛出与捕获
- 异常保护
- 函数异常限定符
- 标准异常类

思考并回答以下问题：

- 异常与Bug的区别
- 程序正式发布时，是否不应有异常保护
- 常见的标准异常类有哪些

熟悉以下代码效果：

- 函数调用种抛出异常时的捕获位置和恢复位置
- 多个catch块的捕获逻辑
- 编写简单的自定义异常类，继承自std::exception

异常 vs Bug

throw 和 try - catch

自定义异常类

Exception

异常 vs Bug

- Bug：程序出错、有**逻辑错误**，无论输入数据是什么都无法得到正确结果

```
float divide(float a, float b) {
    return a * b;
}
```

出错 <-> 正确

- 异常：程序逻辑是正确的，但遇到了**正常范围之外**的输入数据而发生意外中断

```
float divide(float a, float b) {
    return a / b;
}
```

异常 <-> 正常

```
cout << divide(5, 0);
```

Exception

异常 vs Bug

- 对于Bug：程序员可控，越用心进行开发/走查/测试，Bug越少
- 对于Exception：程序员不可控，需要代码中设置针对异常情况的**保护**措施
  - 网络应用：网络连通性、时延等不可控因素对于程序来说是异常
  - 游戏：玩家打出极限操作、突破预先设置的情况是异常
  - 数据分析：数据缺失值、未曾料想的特殊值等情况是异常
  - ……
- 总之，程序正确编译了，也能按照既定逻辑执行功能，但是遇到了合法范围外的数据
- 导致程序失去**确定性**而中断

除以0异常（不除以0就是正常的）

空指针异常（指针不空就是正常的）

数组越界异常（数组不越界就是正常的）

int溢出异常（int不太大就是正常的）

…

异常 vs Bug

throw 和 try - catch

自定义异常类

异常处理 Exception Handle

- 异常导致程序失去确定性，不知道怎么继续下去了
- 异常保护：引导程序继续走下去，再不济也要做好收尾工作，体面结束

如释放占用的内存

try - throw - catch

保护　　抛出　　捕获

- 抛出什么
- 谁来捕获
- 处理完从哪继续

throw 和 try - catch

- throw: 使用throw语句主动立即抛出一个异常**对象**（但不一定throw才抛异常，如n/0）
- try: 保护一个语句块，后跟**一个或多个**catch块。try块中抛出异常不会立刻中止程序而是寻找catch处理
- catch: 捕获**指定**异常并进行处理，处理完**不会再返回**try块，直接离开try-catch继续执行

```
try {
    // 被保护的代码段
    throw(异常对象);
}
catch(期望捕获类型 变量名) {
    // 处理异常的代码段
}
```

- 可以是任何东西
- 但通常是 std::exception 的子类对象

```
try {
    throw(1);
    cout << “After throw”;
}
catch(int e) {
    cout << e;
}
// catch处理完从这里继续
```

不会被执行

会输出1

多个catch块：异常捕获顺序

```
try {
   ...
   throw(something);
   ...
}
catch( Type1 e1 ) {
   // catch 块1
}
catch( Type2 e2 ) {
   // catch 块2
}
...
catch( Type3 eN ) {
   // catch 块N
}
```

- 从上到下依次尝试，直至类型匹配
- 一个异常**只会被捕获一次**

若 A←B←C，哪种catch块排序合理?

```
try { ... }
catch( C e) { ... }
catch( B e) { ... }
catch( A e) { ... }
```

√

```
try { ... }
catch( A e) { ... }
catch( B e) { ... }
catch( C e) { ... }
```

× 不可能捕获到东西

捕获一切异常：catch(...)

```
try {
   throw(something);
}
catch( Type1 e1 ) {
   // catch 块1
}
catch( Type2 e2 ) {
   // catch 块2
}
catch(...) {
  // 处理任何异常
}
```

- catch(...)能捕获一切异常，它后面的其他catch块等于没写
- 因此catch(...)应放在多个catch块的**最后**一个

函数中抛出异常

```
void funcA() {
    try {
        funcB();
    }
    catch (E6 e) { }
    catch (E7 e) { }
    // a处
}
```

最终被这个catch块捕获到

处理完后直接从**a处**继续执行

```
void funcB() {
    try {
        funcC();
    }
    catch (E4 e) { }
    catch (E5 e) { }
    // b处
}
```

```
void funcC() {
    try {
        E6 e;
        throw(e);
    }
    catch (E1 e) { }
    catch (E2 e) { }
    catch (E3 e) { }
    // c处
}
```

在这抛出

函数异常限定符

- 异常限定符：显式地指明是否允许函数抛出(哪些)异常

```
void func1 () throw();
```

什么都不允许抛出

```
void func2 () throw(int,double);
```

允许抛出int或double

```
void func3 () throw(...);
```

允许抛出任何东西

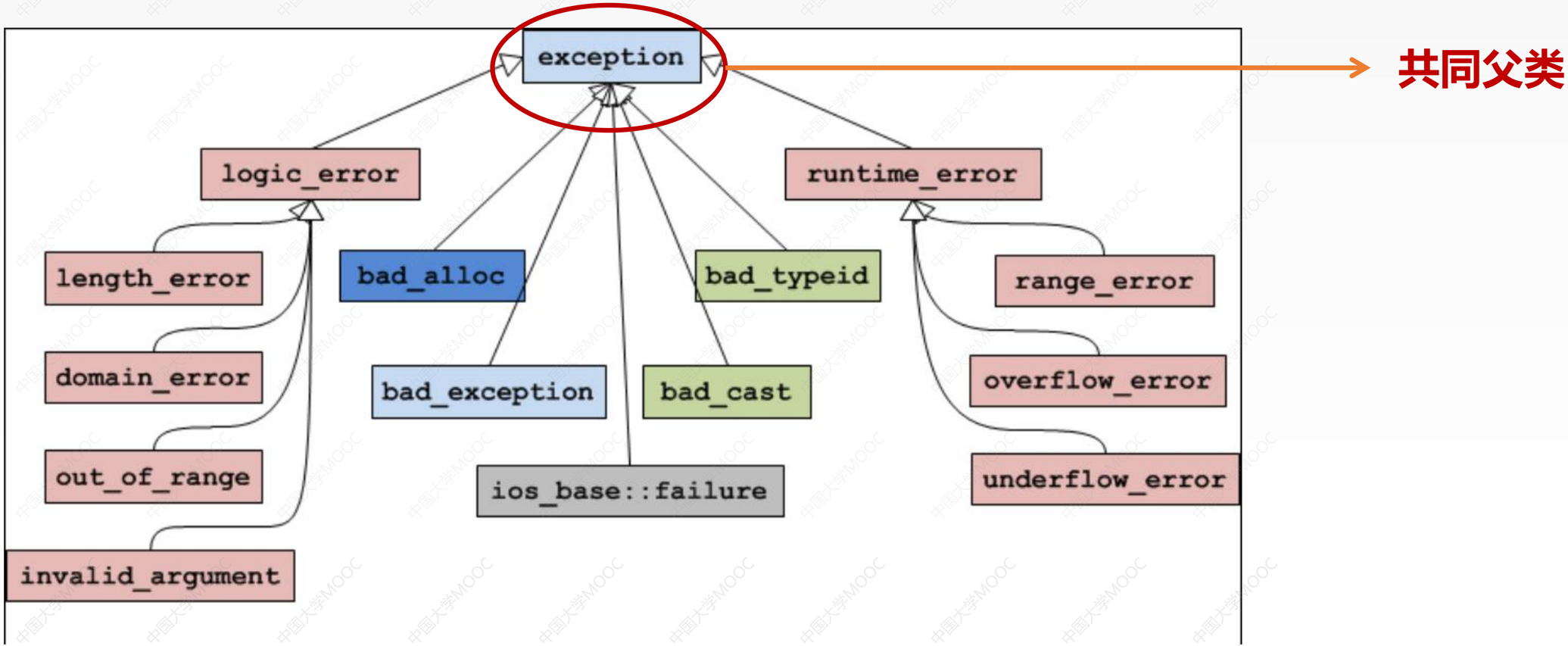
异常 vs Bug

try - throw - catch

自定义异常类

C++标准异常类

- 虽说我们可以手动throw，抛出什么都可以，但一些常见的程序错误是C++自身抛出的（如除以零）
- C++ 根据错误分类根据预设了一系列**标准异常**，定义在 <exception> 中
- 它们以**父子类**层次结构组织，都在名字空间 std 下



自定义异常类

- 自定义异常类通常继承自std::exception，并重写一些方法，比如what()
- 通常在自己编写的程序中会根据业务需求自定义异常类

```
class MyException : public exception {
public:
    string content;
    MyException(string content) {
        this->content = content;
    }
    const char* what() const override{
        return content.c_str();
    }
};
```

```
try {
    MyException e("MyException object");
    throw(e);
    cout << "After throw(e)";
}
catch (MyException& e) {
    cout << e.what();
}
```

自定义异常类例子：三角形异常类

```
class TriangleException : public exception {
public:
    float a;
    float b;
    float c;
    TriangleException(float a, float b, float c) :a(a), b(b), c(c) {}
    const char* what() const override {
        return "Invalid triangle exception!";
    }
};
```

☆ Key 1：try-throw-catch分别用于保护代码段、手动抛出异常与捕获异常

☆ Key 2：抛出的异常会按照其类型被首个匹配的catch块捕获(若有)

1.关于C++的异常处理，正确的说法是：A

A. try必须和catch连用

B. throw必须和catch连用

C. try语句块中必须有throw

D. try-throw-catch这三个关键字必须一起使用

分析：异常对象可以通过throw主动抛出，也可能某处代码执行失败自动抛出(如除以0)

2. 填空：

（1）异常是通过 try 保护 、 throw 抛出 和 catch 捕获 来实现的。

（2）网站后端程序遇到网络不稳定断连是程序 异常 ，误将用户名作为密码进行登录验证是程序 错误

（3）C++程序中，异常处理的主要任务是 处理程序在运行时遇到的问题，尝试恢复运行或善后 .

☆ Key 3：抛出的异常只会在首次匹配的catch块处被捕获一次

☆ Key 4：catch(...){ } 能捕获任何异常

1.假设funcA被调用，则最终将进入哪个catch块，并在处理完后从何处继续执行程序：C

A. ⑤，c处　　B. ④，b处　　C. ①，a处　　D. ①，c处

```
void funcA() {
    try {
        funcB();
    }
    catch (E6 e) {//①}
    catch (E7 e) {//②}
    // a处
}
```

```
void funcB() {
    try {
        funcC();
    }
    catch (E4 e) {//③}
    catch (E5 e) {//④}
    // b处
}
```

```
void funcC() {
    try {
        E6 e;
        throw(e);
    }
    catch (E1 e) {//⑤}
    catch (E2 e) {//⑥}
    catch (E3 e) {//⑦}
    // c处
}
```

☆ Key 3：抛出的异常只会在首次匹配的catch块处被捕获一次

☆ Key 4：catch(...){ } 能捕获任何异常

1.假设funcA被调用，则最终将进入哪个catch块，并在处理完后从何处继续执行程序：B

A. ⑤，c处　　B. ④，b处　　C. ①，a处　　D. ①，c处

```
void funcA() {
    try {
        funcB();
    }
    catch (E6 e) {//①}
    catch (E7 e) {//②}
    // a处
}
```

```
void funcB() {
    try {
        funcC();
    }
    catch (E4 e) {//③}
    catch (...) {//④}
    // b处
}
```

```
void funcC() {
    try {
        E6 e;
        throw(e);
    }
    catch (E1 e) {//⑤}
    catch (E2 e) {//⑥}
    catch (E3 e) {//⑦}
    // c处
}
```

中国大学MOOC
搜索：C++不挂科