类型转换运算符、异常、IO

类型转换运算符

- 使用 C 风格的类型转换可以把想要的任何东西转换成我们需要的类型，但是这种类型转换太过松散，对于这种松散的情况，C++ 提供了更严格的类型转换，可以提供更好的控制转换过程，并添加 4 个类型转换运算符，使转换过程更规范：
  - static_cast
  - dynamic_cast
  - const_cast
  - reinterpret_cast
- 可以根据目的选择一个合适的运算符，而不是使用通用的类型转换。这指出了进行类型转换的原因，并让编译器能够检查程序的行为是否与设计者想法吻合。

静态转换（static_cast）

- 用于类层次结构中基类（父类）和派生类（子类）之间指针或引用的转换
  - 进行上行转换（把派生类的指针或引用转换成基类表示）是安全的
  - 进行下行转换（把基类指针或引用转换成派生类表示）时，由于没有动态类型检查，所以是不安全的
- 用于基本数据类型之间的转换，如把 `int` 转换成 `char`，把 `char` 转换成 `int`。这种转换的安全性也要开发人员来保证
- `static_cast< type-name > (expression)`

动态转换（dynamic_cast）

- dynamic_cast 主要用于类层次间的上行转换和下行转换
- 在类层次间进行上行转换时，dynamic_cast 和 static_cast 的效果是一样的
- 在进行下行转换时，dynamic_cast 具有类型检查的功能，比 static_cast 更安全
- dynamic_cast< type-name > (expression)

常量转换（const_cast）

- 该运算符用来修改类型的const属性
- 常量指针被转化成非常量指针，并且仍然指向原来的对象
- 常量引用被转换成非常量引用，并且仍然指向原来的对象
- `const_cast< type-name > (expression)`
- 注意:不能直接对非指针和非引用的变量使用 `const_cast` 操作符

重新解释转换（reinterpret_cast）

- 这是最不安全的一种转换机制，最有可能出问题
- 主要用于将一种数据类型从一种类型转换为另一种类型，它可以将一个指针转换成一个整数，也可以将一个整数转换成一个指针
- `reinterpret_cast< type-name > (expression)`

异常概述

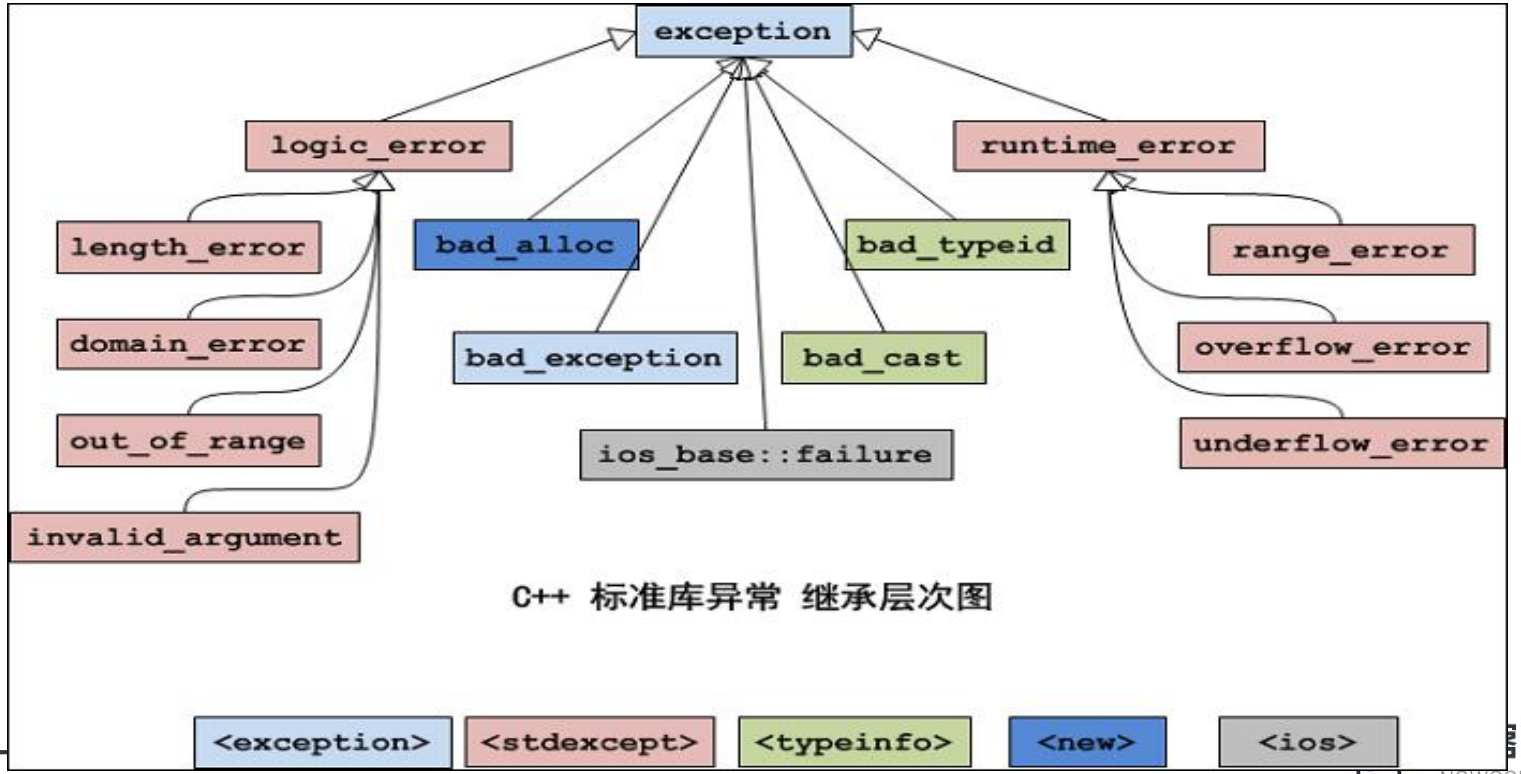
- 程序有时会遇到运行阶段错误，导致程序无法正常运行下去。（如：除 0 溢出、数组下标越界、读取的文件不存在、空指针、内存不足等），C++ 异常为处理这种情况提供了一种功能强大而灵活的工具。
- C 语言中如何处理错误？
  - 使用整型的返回值标识错误
  - 二是使用 errno 宏（可以简单的理解为一个全局整型变量）去记录错误

C++ 异常机制相比 C 语言异常处理的优势

- 函数的返回值可以忽略，但异常不可忽略。如果程序出现异常，但是没有被捕获，程序就会终止，这多少会促使程序员开发出来的程序更健壮一点。而如果使用 C 语言的 error 宏或者函数返回值，调用者都有可能忘记检查，从而没有对错误进行处理，结果造成程序莫名终止或出现错误的结果。
- 整型返回值没有任何语义信息。而异常却包含语义信息，有时从类名就能够体现出来。
- 整型返回值缺乏相关的上下文信息。异常作为一个类，可以拥有自己的成员，这些成员就可以传递足够的信息。
- 异常处理可以在调用跳级。这是一个代码编写时的问题：假设在有多个函数的调用栈中出现了某个错误，使用整型返回码要求你在每一级函数中都要进行处理。而使用异常处理的栈展开机制，只需要在一处进行处理就可以了，不需要每级函数都处理。

C++标准异常库

- 标准库中也提供了很多的异常类，它们是通过类继承组织起来的。



标准异常类的成员

- 在上述继承体系中，每个类都有提供了构造函数、复制构造函数、和赋值操作符重载。
- 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
- 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。

标准异常类的具体描述

异常名称	描述
exception	所有标准异常类的父类
bad_alloc	当 operator new and operator new[]，请求分配内存失败时
bad_exception	这是个特殊的异常，如果函数的异常抛出列表里声明了 bad_exception 异常，当函数内部抛出了异常抛出列表中没有的异常，就是调用的 unexpected 函数中若抛出异常，不论什么类型，都会被替换为 bad_exception 类型
bad_typeid	使用 typeid 操作符，操作一个 NULL 指针，而该指针是带有虚函数的类，这时抛出 bad_typeid异常
bad_cast	使用 dynamic_cast 转换引用失败的时候
ios_base::failure	IO 操作过程出现错误
logic_error	逻辑错误，可以在运行前检测的错误
runtime_error	运行时错误，仅在运行时才可以检测的错误

logic_error的子类

异常名称	描述
length_error	试图生成一个超出该类型最大长度的对象时，例如 vector 的 resize操作
domain_error	参数的值域错误，主要用在数学函数中。例如使用一个负值调用只能操作非负数的函数
out_of_range	超出有效范围
invalid_argument	参数不合适。在标准库中，当利用 string 对象构造 bitset 时，而 string 中的字符不是'0'或'1'的时候，抛出该异常

runtime_error的子类

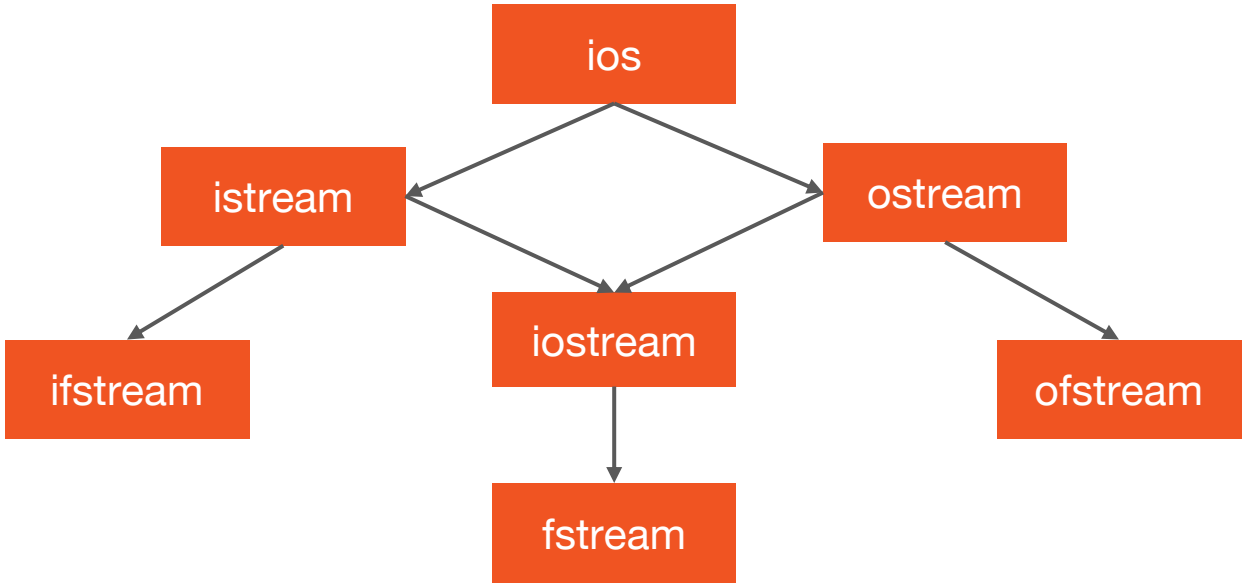
异常名称	描述
range_error	计算结果超出了有意义的值域范围
overflow_error	算术计算上溢
underflow_error	算术计算下溢

I/O流概述

- C++ 输入输出包含以下三个方面的内容：
  - 对系统指定的标准设备的输入和输出。即从键盘输入数据，输出到显示器屏幕。这种输入输出称为标准的输入输出，简称标准I/O。
  - 以外存磁盘文件为对象进行输入和输出，即从磁盘文件输入数据，数据输出到磁盘文件。以外存文件为对象的输入输出称为文件的输入输出，简称文件I/O。
  - 对内存中指定的空间进行输入和输出。通常指定一个字符数组作为存储空间(实际上可以利用该空间存储任何信息)。这种输入和输出称为字符串输入输出，简称串I/O。

I/O流概述

- C++ 提供了用于输入输出的 iostream 类库。iostream 这个单词是由3个部分组成的，即 i-o-stream，意为输入输出流。在 iostream 类库中包含许多用于输入输出的类。



I/O类库中的常见流类

类名	作用	头文件
ios	抽象基类	iostream
istream ostream iostream	通用输入流和其他输入流的基类 通用输出流和其他输出流的基类 通用输入输出流和其他输入输出流的基类	iostream
ifstream ofstream fstream	输入文件流类 输出文件流类 输入输出文件流类	fstream
istrstream ostrstream strstream	输入字符串流类 输出字符串流类 输入输出字符串流类	strstream

标准 I/O 对象

- 在 iostream 头文件中定义了 4 种标准流对象：cin、cout、cerr、clog
  - cout 是 console output 的缩写，意为在控制台（终端显示器）的输出；
  - cerr 流对象是标准错误流，cerr 流已被指定为与显示器关联。cerr 的作用是向标准错误设备(standard error device)输出有关出错信息；
  - clog 流对象也是标准错误流，它是 console log 的缩写。它的作用和 cerr 相同，都是在终端显示器上显示出错信息。区别：cerr 是不经过缓冲区，直接向显示器上输出有关信息，而 clog 中的信息存放在缓冲区中，缓冲区满后或遇 endl 时向显示器输出。

控制输出的流成员函数

流成员函数	与之作用相同的控制符	作用
precision(n)	setpresion(n)	设置实数的精度为n
width(n)	setw(n)	设置字段宽度为n位
fill(c)	setfill(c)	设置填充字符c
setf()	setiosflags()	设置输出格式状态，括号中应给出格式状态，内容与控制符setiosflags括号中的内容相同
unsetf()	resetiosflags()	终止已设置的输出格式状态，在括号中应指定内容

设置格式状态的格式标志

格式标志	作用
ios::left	输出数据在本域宽范围内向左对齐
ios::right	输出数据在本域宽范围内向右对齐
ios::internal	数值的符号位在域宽内左对齐，数值右对齐，中间由填充字符填充
ios::dec	设置整数的基数为10
ios::oct	设置整数的基数为8
ios::hex	设置整数的基数为16
ios::showbase	强制输出整数的基础（八进制数以0开头，十六进制数以0x开头）

设置格式状态的格式标志

格式标志	作用
ios::showpoint	强制输出浮点数的小点和尾数0
ios::uppercase	在以科学计数法格式E和以十六进制输出字母时以大写表示
ios::showpos	对整数显示“+”号
ios::scientific	浮点数以科学计数法格式输出
ios::fixed	浮点数以定点格式（小数形式）输出
ios::unitbuf	每次输出之后刷新所有的流
ios::stdio	每次输出之后清除stdout，stderr

输入输出流的控制符

控制符	作用
dec	设置数值的基数为10
hex	设置数值的基数为16
oct	设置数值的基数为8
setfill(c)	设置填充字符c，c可以是字符常量或字符变量
setprecision(n)	设置浮点数的精度为n位。在以一般十进制小数形式输出时，n代表有效数字。在以fixed（固定位小数）形式和scientific（指数）形式输出时，n为小数位数
setw(n)	设置字段宽度为n
setiosflags(ios::fixed)	设置浮点数以固定的小数位数显示

输入输出流的控制符

控制符	作用
setiosflags(ios::scientific)	设置浮点数以科学计数法（即指数形式）显示
setiosflags(ios::left)	输出数据左对齐
setiosflags(ios::right)	输出数据右对齐
setiosflags(ios::skipws)	忽略前导的空格
setiosflags(ios::uppercase)	数据以十六进制形式输出的字母以大写表示
setiosflags(ios::lowercase)	数据以十六进制形式输出的字母以小写表示
setiosflags(ios::showpos)	输出整数时给出“+”号

注意：如果使用了控制符，在程序单位的开头除了要加iostream头文件外，还要加iomanip头文件

文件流

- 输入输出是以系统指定的标准设备（输入设备为键盘，输出设备为显示器）为对象的。在实际应用中，常以磁盘文件作为对象。即从磁盘文件读取数据，将数据输出到磁盘文件。
- 和文件有关系的输入输出类主要在 `fstream.h` 这个头文件中被定义，在这个头文件中主要被定义了三个类，由这三个类控制对文件的各种输入输出操作，他们分别是 `ifstream`、`ofstream`、`fstream`，其中 `fstream` 类是由 `iostream` 类派生而来。
- 由于文件设备并不像显示器屏幕与键盘那样是标准默认设备，所以它在 `fstream` 头文件中是没有像 `cout` 那样预先定义的全局对象，所以我们必须自己定义一个该类的对象。

文件输入输出方式设置值

方式	作用
ios::in	以输入方式打开文件
ios::out	以输出方式打开文件（这是默认方式），如果已有此名字的文件，则将其原有内容全部清除
ios::app	以输出方式打开文件，写入的数据添加在文件末尾
ios::ate	打开一个已有的文件，文件指针指向文件末尾
ios::trunc	打开一个文件，如果文件已经存在，则删除其中全部数据，如果文件不存在，则建立新文件。如已指定了ios::out方式，而未指定ios::app，ios::ate，ios::in，则同时默认此方式
ios::binary	以二进制方式打开一个文件，如不指定此方式则默认为ASCII方式

文件输入输出方式设置值

方式	作用
ios::nocreate	打开一个已有的文件，如文件不存在，则打开失败。nocreate的意思是不建立新文件
ios::noreplace	如果文件不存在则建立新文件，如果文件已存在则操作失败，replace的意思是不更新原有文件
ios::in | ios::out	以输出和输出方式打开文件，文件可读可写
ios::out | ios::binary	以二进制方式打开一个输出文件
ios::in | ios::binary	以二进制方式打开一个输入文件

Thanks