Modern cpp notes 30 lectures — exception

Friday, December 18, 2020 4:02 PM

不用异常的例子(in C):

```
typedef struct{
   float *  data;
   size_t   nrows;
   size_t   ncols;
}matrix;

enum Matrix_err_code{
    MATRIX_SUCCESS,
    MATRIX_ERR_MEMORY_INSUFFICIENT,
    ...
};

int matrix_alloc(matrix*  ptr,
                 size_t  nrows,
                 size_t  ncols)
{
  size_t size = nrows * ncols * sizeof(float);
  float* data = malloc(size);
  if (data == NULL){
     return MATRIX_ERR_MEMORY_INSUFFICIENT;
  }
  ptr->data = data;
  ptr -> nrows = nrows;
  ptr -> ncols = ncols;
}

void matrix_dealloc(matrix* ptr)
{
  if(ptr->data == NULL) {
     return;
  }
  free(ptr->data);
  ptr -> data = NULL;
  ptr-> nrows = 0;
  ptr -> n cols = 0;
}

int matrix_multiply(Matrix*  result,
                    const  matrix * lhs,
                    const  matrix * rhs)
{
  int errcode;
  if(lhs -> ncols != rhs -> nrows) {
     return  MATRIX_ERR_MISMATCHED_MATRIX_SIZE;
  }

  errcode = matrix_alloc(
     result, lhs->nrows, rhs->ncols);
  if(errcode != MATRIX_SUCCESS) {
      return errcode;
  }

  return  MATRIX_SUCCESS;
}
```

调用代码:

```
matrix c;

memset(c, 0, sizeof(matrix));

errcode = matrix_Multiply(c, a, b);
if (errcode != MATRIX_SUCCESS) {
    goto error_exit;
}

error_exit:
  matrix_dealloc(&c);
  return errcode;
```

可以看到,有大量需要判断错误的代码,零散分布在各处.

使用异常,就可以在构造函数里做真正的初始化工作了.

矩阵类有下列的数据成员:

```
class matrix {
  ...
 private:
   float*  data_;
   size_t  nrows_;
   size_t  ncols_;
}
```

构造函数

```
matrix::matrix(size_t  nrows,
               size_t  ncols)
{
  data_ = new float[nrows * ncols];
  nrows_ = nrows;
  ncols_ = ncols;
}
```

析构

```
matrix::~Matrix()
{
  delete[] data_;
}
```

delete[] operator deallocates memory and calls destructors for an array of objects created with new[].

delete operator deallocates memory and calls the destructor for a single object created with new.

乘法函数

```
class matrix{
  ...
  friend matrix
  operator*(const matrix&,
            const matrix&);
};

matrix operator*(const matrix& lhs,
                 const matrix& rhs)
{
  if (lhs.ncols != rhs.nrows){
    throw std::runtime_error("Matrix sizes mismatch");
  }

  matrix result(lhs.nrows, rhs.ncols);
  return result;
}
```

异常处理只有一个throw, 没有try-catch.

使用乘法

```
Matrix  c =  a*b;
```

△ 异常安全:

当异常发生时,既不会发生资源泄漏,系统也不会处于一个不一致的状态.

可能会出现异常的地方

- 内存分配. 如果new 出错, 会报 bad_alloc. 对象析构失败. 所有的栈上对象会全部被析构, 资源被清理.
- 矩阵长宽不合适, 不能做乘法, 产生异常, 对象 c 不会被创造.
- 乘法函数里内存分配失败, result对象没有构造出来, 也没有 c 对象.
- 如果a, b是本地变量, 乘法失败时, 析构函数会自动释放其空间, 不会有任何资源泄漏.

从C++17开始, 无法在函数里声明可能抛出异常. 唯一能声明的, 就是函数不会抛出异常, e.g. noexcept, noexcept(true), throw().

如果一个函数声明了不会抛出异常却抛出了异常, C++会调用 std::terminate 终止程序.

类的特殊成员, 如构造函数, 析构函数, 赋值函数等会自动成为 noexcept.

C++的标准容器提供了at成员函数, 能够在下标不存在的时候抛出异常.

```
#include<iostream> //std::cout/endl
#include <stdexcept> // std::out_of_range
#include <vector>  // std::vector

vector<int>  v{1, 2, 3};
v[0]; ← 1
v.at(0); ← 1.
v[3]; ← -1342175236
try{
  v.at(3);
}
catch(const out_of_range& e){
   cerr << e.what() << endl;
}
```

_M_range_check: __n (which is 3) >= this->size() (which is 3).

C++的标准容器在大部分情况下提供了强异常保证, 即一旦发生异常, 现场会恢复到调用函数之前的状态, 容器的内容不会发生改变, 也没有任何资源泄漏. Eg. vector会在元素类型没有提供保证不抛异常的移动构造函数情况下, 在移动元素的时候会使用拷贝构造函数. 因为一旦操作异常, 被移动的元素已破坏, 处于只能析构的状态, 异常安全性就得不到保证.