UNIT 12 异常处理

武汉大学计算机学院程序设计课程组
主讲人：常 军

E-MAIL：chunscc@163.com
电 话：18986211771
QQ 群: 1020712774

本讲提纲

- 异常处理的基本思想
- C++异常处理的实现
- 异常处理中的构造与析构
- 标准程序库异常处理
- 深度探索

1. 异常处理的基本思想

异常处理的基本思想

没有进行错误处理的程序:

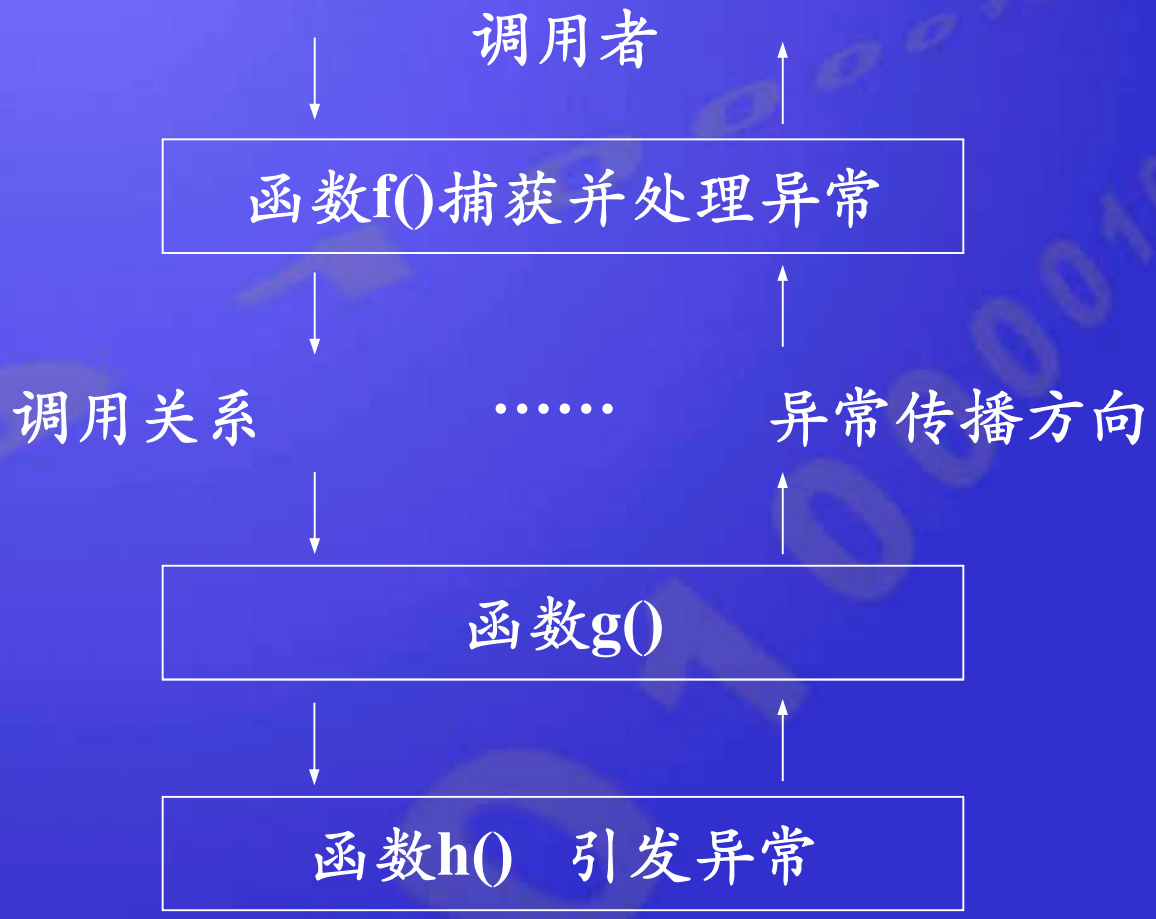
```
{   openTheFile;
  determine its size;
  allocate that much memory;
  read-file
  closeTheFile;
}
```

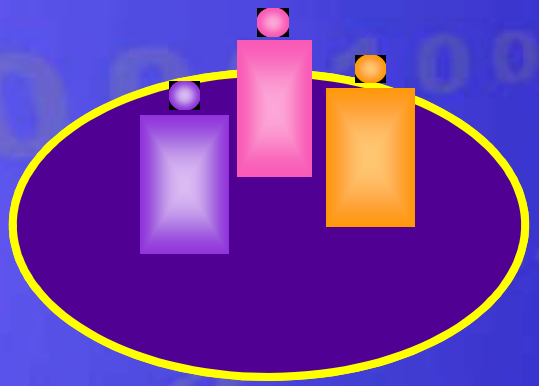
- 观察这个程序，会发现大部分精力花在出错处理上
- 只能考虑到部分错误，对其它的情况无法处理
- 程序可读性差
- 出错返回信息量太少

以常规方法进行错误处理：

```
openFiles;
if (theFilesOpened)  {
   determine the length of the file;
   if (gotTheFileLength)   {
      allocate that much memory;
      if (gotEnoughMemory)       {
           read the file into memory;
           if (readFailed)  errorCode=-1;
           else errorCode = -2;
      }
      else  errorCode=-3;
   }
   else errorCode=-4 ;
}
else errorCode=-5;
```

12.1 异常处理的基本思想





2. C++异常处理的实现

12.2.1 异常处理的语法

抛掷异常的程序段

```
.....
throw 表达式;
.....
```

捕获并处理异常的程序段

```
try                                  保护段
   复合语句
catch（异常声明）                    异常处理程序
   复合语句
catch（异常声明）
   复合语句
   …
```

12.2.1 异常处理的语法（续）

若有异常则通过throw操作创建一个异常对象并抛掷。

将可能抛出异常的程序段嵌在try块之中。控制通过正常的顺序执行到达try语句，然后执行try块内的保护段。

如果在保护段执行期间没有引起异常，那么跟在try块后的catch子句就不执行。程序从try块后跟随的最后一个catch子句后面的语句继续执行下去。

catch子句按其在try块后出现的顺序被检查。匹配的catch子句将捕获并处理异常（或继续抛掷异常）。

如果匹配的处理器未找到，则运行库函数terminate将被自动调用，其缺省功能是调用abort终止程序。

例12-1处理除零异常

```
//12_1.cpp
#include <iostream>
using namespace std;
int divide(int x, int y) {
        if (y == 0)
                throw x;
        return x / y;
}
int main() {
        try {
                cout << "5 / 2 = " << divide(5, 2) << endl;
                cout << "8 / 0 = " << divide(8, 0) << endl;
                cout << "7 / 1 = " << divide(7, 1) << endl;
        } catch (int e) {
                cout << e << " is divided by zero!" << endl;
        }
        cout << "That is ok." << endl;
        return 0;
}
```

结果如下：

```
5 / 2 = 2
8 is divided by zero!
That is ok.
```

12.2.2 异常接口声明

可以在函数的声明中列出这个函数可能抛掷的所有异常类型。

例如：

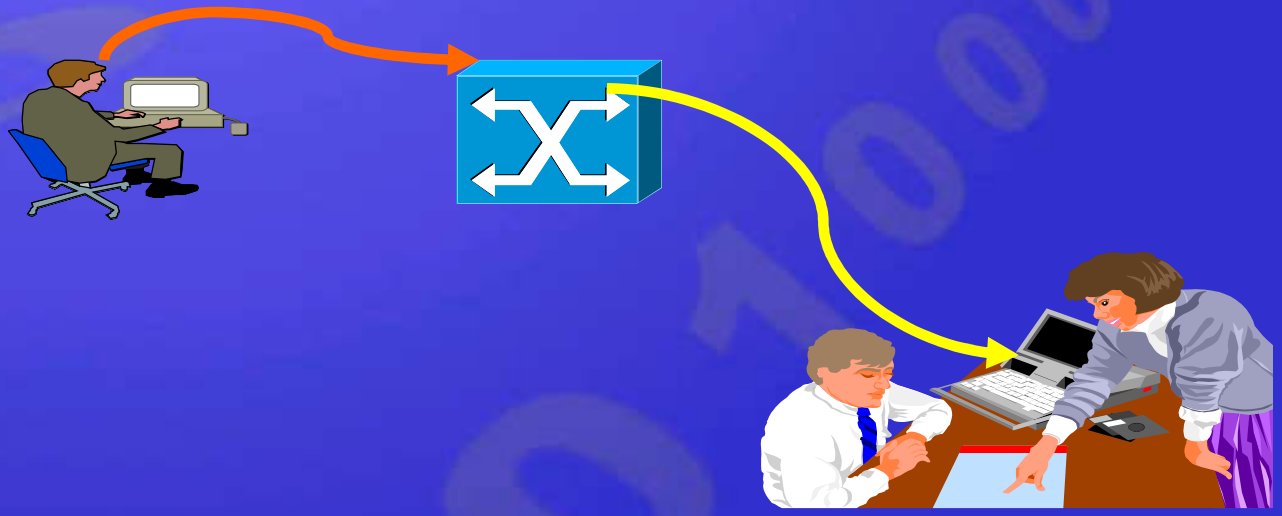
void fun() throw(A，B，C，D);

若无异常接口声明，则此函数可以抛掷任何类型的异常。

不抛掷任何类型异常的函数声明如下：

void fun() throw();

3. 异常处理中的构造和析构



12.3 异常处理中的构造与析构

找到一个匹配的catch异常处理后

- ✓ 初始化异常参数。
- ✓ 将从对应的try块开始到异常被抛掷处之间构造（且尚未析构）的所有自动对象进行析构。
- ✓ 从最后一个catch处理之后开始恢复执行。

例12-2使用带析构语义的类的C++异常处理

```
//12_2.cpp
#include <iostream>
#include <string>
using namespace std;
class MyException {
public:
        MyException(const string &message) : message(message) {}
        ~MyException() {}
        const string &getMessage() const { return message; }
private:
        string message;
};

class Demo {
public:
        Demo() { cout << "Constructor of Demo" << endl; }
        ~Demo() { cout << "Destructor of Demo" << endl; }
};
```

```
void func() throw (MyException) {
        Demo d;
        cout << "Throw MyException in func()" << endl;
        throw MyException("exception thrown by func()");
}

int main() {
        cout << "In main function" << endl;
        try {
                func();
        } catch (MyException& e) {
                cout << "Caught an exception: " << e.getMessage() << endl;
        }
        cout << "Resume the execution of main()" << endl;
        return 0;
}
```

例12-2（续）

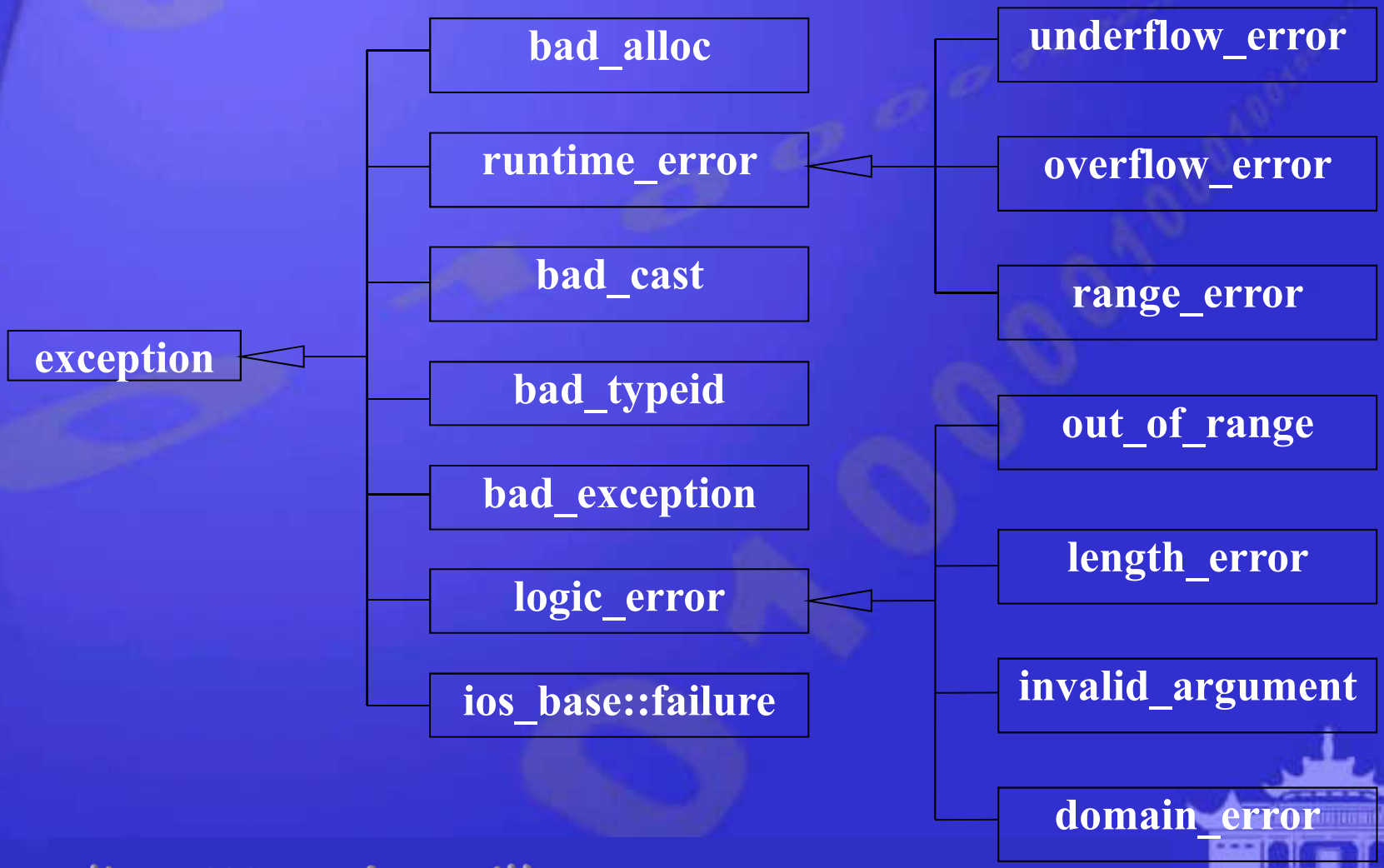
例12-2（续）

结果如下:

```
In main function
Constructor of Demo
Throw MyException in func()
Destructor of Demo
Caught an exception: exception thrown by func()
Resume the execution of main()
```


4. 标准程序异常处理

12.4 标准程序库异常处理



C++标准库各种异常类所代表的异常

异常类	头文件	异常的含义
bad_alloc	exception	用new动态分配空间失败
bad_cast	new	执行dynamic_cast失败（dynamic_cast参见8.7.2节）
bad_typeid	typeinfo	对某个空指针p执行typeid(*p)（typeid参见8.7.2节）
bad_exception	typeinfo	当某个函数fun()因在执行过程中抛出了异常声明所不允许的异常而调用unexpected()函数时，若unexpected()函数又一次抛出了fun()的异常声明所不允许的异常，且fun()的异常声明列表中有bad_exception，则会有一个bad_exception异常在fun()的调用点被抛出
ios_base::failure	ios	用来表示C++的输入输出流执行过程中发生的错误
underflow_error	stdexcept	算术运算时向下溢出
overflow_error	stdexcept	算术运算时向上溢出
range_error	stdexcept	内部计算时发生作用域的错误
out_of_range	stdexcept	表示一个参数值不在允许的范围之内
length_error	stdexcept	尝试创建一个长度超过最大允许值的对象
invalid_argument	stdexcept	表示向函数传入无效参数
domain_error	stdexcept	执行一段程序所需要的先决条件不满足

标准异常类的基础

exception：标准程序库异常类的公共基类

logic_error表示可以在程序中被预先检测到的异常

- ✓ 如果小心地编写程序，这类异常能够避免

runtime_error表示难以被预先检测的异常

例12-3 三角形面积计算

编写一个计算三角形面积的函数，函数的参数为三角形三边边长a、b、c，可以用Heron公式计算：

设 $p=\frac{a+b+c}{2}$ ，则三角形面积 $S=\sqrt{p(p-a)(p-b)(p-c)}$

例12-3（续）

```
//12_3.cpp
#include <iostream>
#include <cmath>
#include <stdexcept>
using namespace std;
//给出三角形三边长，计算三角形面积
double area(double a, double b, double c)  throw (invalid_argument) {
//判断三角形边长是否为正
        if (a <= 0 || b <= 0 || c <= 0)
                throw invalid_argument("the side length should be positive");
//判断三边长是否满足三角不等式
        if (a + b <= c || b + c <= a || c + a <= b)
   throw invalid_argument("the side length should fit the triangle inequation");
//由Heron公式计算三角形面积
        double s = (a + b + c) / 2;
        return sqrt(s * (s - a) * (s - b) * (s - c));
}
```

例12-3（续）

```
int main() {
        double a, b, c;    //三角形三边长
        cout << "Please input the side lengths of a triangle: ";
        cin >> a >> b >> c;
        try {
                double s = area(a, b, c);  //尝试计算三角形面积
                cout << "Area: " << s << endl;
        } catch (exception &e) {
                cout << "Error: " << e.what() << endl;
        }
        return 0;
}
```

例12-3（续）

运行结果1:

```
Please input the side lengths of a triangle: 3 4 5
Area: 6
```

运行结果2:

```
Please input the side lengths of a triangle: 0 5 5
Error: the side length should be positive
```

运行结果2:

```
Please input the side lengths of a triangle: 1 2 4
Error: the side length should fit the triangle inequation
```


5. 综合实例

——对个人银行账户管理程序的改进

12.5 综合实例

——对个人银行账户管理程序的改进

本例中，在构造或输入一个Date对象时如发生了错误，直接使用标准程序库中的runtime_error构造异常并抛出；在账户类中如发生了错误，由于希望异常信息能够标识是哪个账户发生了错误。

本程序中创建了一个类AccountException，该类从runtime_error派生，该类中保存了一个Account型常指针，指向发生错误的账户，这样在主函数中，输出错误信息的同时也可以将账号输出。

例12-4

```
//date.cpp，仅列出与以前不同的内容，下同
#include "date.h"
#include <iostream>
#include <stdexcept>
using namespace std;
Date::Date(int year, int month, int day) : year(year), month(month), day(day) {
        if (day <= 0 || day > getMaxDay())
    throw runtime_error("Invalid date");
        int years = year - 1;
        totalDays = years * 365 + years / 4 - years / 100 + years / 400 +
DAYS_BEFORE_MONTH[month - 1] + day;
        if (isLeapYear() && month > 2) totalDays++;
}
istream & operator >> (istream &in, Date &date) {
        int year, month, day;
        char c1, c2;
        in >> year >> c1 >> month >> c2 >> day;
        if (c1 != '-' || c2 != '-')
                throw runtime_error("Bad time format");
        date = Date(year, month, day);
        return in;
}
```

例12-4（续）

```
//account.h
#ifndef __ACCOUNT_H__
#define __ACCOUNT_H__
#include "date.h"
#include "accumulator.h"
#include <string>
#include <map>
#include <istream>
#include <stdexcept>
//account.h中增加了以下类，其它各类的定义与例11-13完全相同，不再重复给出
class AccountException : public std::runtime_error {
private:
        const Account *account;
public:
        AccountException(const Account *account, const std::string &msg)
                : runtime_error(msg), account(account) { }
        const Account *getAccount() const { return account; }
};
#endif //__ACCOUNT_H__
```

例12-4（续）

```
//account.cpp中仅以下成员函数的实现与例11-13不同，其它内容皆与之完全相同
void Account::error(const string &msg) const {
        throw AccountException(this, msg);
}
//12_4.cpp仅主函数的实现与例11_13.cpp不同，其它皆与之完全相同
int main() {
        Date date(2008, 11, 1);        //起始日期
        Controller controller(date);
        string cmdLine;
        const char *FILE_NAME = "commands.txt";
        ifstream fileIn(FILE_NAME);          //以读模式打开文件
        if (fileIn) {          //如果正常打开，就执行文件中的每一条命令
                while (getline(fileIn, cmdLine)) {
                        try {
                                controller.runCommand(cmdLine);
                        } catch (exception &e) {
                                cout << "Bad line in " << FILE_NAME << ":
    " << cmdLine << endl;
                                cout << "Error: " << e.what() << endl;
                                return 1;
                        }
                }
                fileIn.close();        //关闭文件
        }
```

```
 ofstream fileOut(FILE_NAME, ios_base::app);   //以追加模式
        cout << "(a)add account (d)deposit (w)withdraw (s)show (c)change day
(n)next month (q)query (e)exit" << endl;
        while (!controller.isEnd()) {//从标准输入读入命令并执行，直到退出
                cout    <<    controller.getDate()    <<    "\tTotal:    "    <<
Account::getTotal()
                        << "\tcommand> ";
                string cmdLine;
                getline(cin, cmdLine);
                try {
                        if (controller.runCommand(cmdLine))
                           fileOut << cmdLine << endl;    //将命令写入文件
                } catch (AccountException &e) {
                   cout <<"Error(#" <<e.getAccount()->getId() << "): "
                         << e.what() << endl;
                } catch (exception &e) {
                        cout << "Error: " << e.what() << endl;
                }
        }
        return 0;
}
```

例12-4（续）

例12-5（续）

运行结果如下：

```
.....（前面的输入和输出与例9-16给出的完全相同，篇幅所限，不再重复）
2009-1-1      Total: 20482.9  command> w 2 20000 buy a car
Error(#C5392394): not enough credit
2009-1-1      Total: 20482.9  command> w 2 1500 buy a television
2009-1-1      #C5392394      -1500   -1550    buy a television
2009-1-1      Total: 18982.9  command> q 2008-12-5 2009-1-32
Error: Invalid date
2009-1-1      Total: 18982.9  command> q 2008-12-5 2009-1-31
2008-12-5     #S3755217      5500    10500    salary
2009-1-1      #S3755217      17.77   10517.8  interest
2009-1-1      #02342342      15.16   10015.2  interest
2009-1-1      #C5392394      -50     -50      annual fee
2009-1-1      #C5392394      -1500   -1550    buy a television
2009-1-1      Total: 18982.9  command> e
```

6. 深度探索

12.6.1 异常安全性问题

一个异常安全的函数，在有异常抛出时：

- ✓ 不应泄露任何资源
- ✓ 不能使任何对象进入非法状态

反例：例9-8中的下列代码：

```
template <class T, int SIZE>
void Stack<T, SIZE>::push(const T &item) {
        assert(!isFull()); //如果栈满了，则报错
        list[++top] = item;          //将新元素压入栈顶
}
```

如果赋值过程中有异常抛出，由于top已经增1，栈顶的内容将变得不确定。

12.6.1 异常安全性问题（续）

该函数的修正版本:

```
template <class T, int SIZE>
void Stack<T, SIZE>::push(const T &item) {
        assert(!isFull()); //如果栈满了，则报错
        list[top + 1] = item;//将新元素压入栈顶
        top++;
}
```

即使赋值时抛出异常，由于此时top并没有真正增1，因此当前对象的状态没有改变，该函数是异常安全的。

编写异常安全程序的原则

明确哪些操作绝对不会抛掷异常

- ✓ 这些操作是异常安全编程的基石
- ✓ 例：基本数据类型的绝大部分操作，指针的赋值、算术运算和比较运算，STL容器的swap函数

尽量确保析构函数不抛掷异常

12.6.2 避免异常发生时的资源泄漏

一个函数，必须在有异常向外抛出前，释放应由它负责释放的资源。

通常的解决方案

- ✓ 把一切动态分配的资源都包装成栈上的对象，利用抛掷异常时自动调用对象析构函数的特性来释放资源。
- ✓ 对于必须在堆上构造的对象，可以用智能指针**auto_ptr**加以包装。

智能指针auto_ptr

C++标准库的一个类模板

- ✓ 在memory头文件中定义
- ✓ 有一个类型参数X，表示智能指针指向数据的类型
- ✓ 每个智能指针对象关联一个普通指针

构造函数：

explicit auto_ptr(X *p = 0) throw();

获得与智能指针对象关联的指针：**X *get() const throw();**

- ✓ 由于auto_ptr的“*”与“->”运算符已被重载，对一个auto_ptr的对象使用“*”和“->”，等价于对它所关联的指针使用相应运算符。

智能指针auto_ptr（续）

更改智能指针对象关联的指针

void reset(X *p = 0) throw();

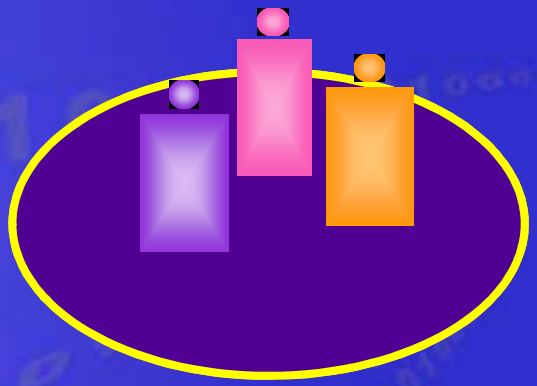
原指针所指堆对象会被删除

解除与当前指针的关联

X* release() throw();

注意事项

- ✓ 智能指针对象执行赋值和拷贝构造时，原对象的关联指针会被解除



本讲小结

本讲主要知识点

主要内容

1. 异常处理的基本思想
2. C++异常处理的实现
3. 异常处理中的构造与析构

达到的目标：简单了解C++的异常处理机制

第12讲上机练习

学生用书，实验12，异常处理

第12讲 课后练习

作业：

12-4 设计一个异常 Exception 抽象类，在此基础上派生一个OutOfMemory类响应内存不足，一个RangeError类响应输入的数不在指定范围内，实现并测试这几个类。

12—5 在程序中用 new 分配内存时，如果操作未成功，则用 try 语句触发一个字符型异常，用 catch 语句捕获此异常。

12—6 定义一个异常类 CException，有成员函数 Reason（），用来显示异常的类型，定义函数fn1（）触发异常，在主函数的try模块中调用fn1（），在catch模块中捕获异常，观察程序的执行流程。

本 讲 结 束