

Ch-14

C++工具

主要内容

□ 异常处理

□ 命名空间

□ 使用C的标准函数库

14.1 异常处理

14.1.1 异常处理的任务

异常处理是对运行时出现的差错以及其他非正常情况的处理。

程序中如果没有异常处理机制，一旦运行时出现异常，则程序本身不能处理，导致运行终止；但如果设置了异常处理机制，则出现异常后，程序流程会转到用户指定的异常处理代码段去处理异常。

14.1.2 异常处理的方法

C++的异常处理机制包括3个部分：try、throw和catch。

其中：

- 1. 在try部分包含可能产生异常的一条或多条语句；**
- 2. 如果发现异常，则在throw部分发出一条异常信息（通常都是一个错误）；**
- 3. 最后catch部分捕捉到发出的异常信息，并按照用户指定的操作处理异常。**
 - 这里可以有多个catch实现，每个catch必须指定一个它能够处理的异常类型。**

例14.1 异常处理1

说明：（1）执行throw语句后，流程立即离开triangle函数，转到上一级函数中的catch部分进行执行；（2）throw后面可以是任意类型的数据；（3）throw后会查找匹配的catch子句，此例中的a是double型，而catch子句执行的异常信息也是double型，因此二者匹配，则执行catch子句中的异常处理；（4）异常处理后将继续执行catch子句后面的语句。

```
5  int main()
6  {
7      double triangle(double, double, double);
8      double a, b, c;
9      cin >> a >> b >> c;
10     try {
11         while(a>0 && b>0 && c>0){
12             cout << triangle(a, b, c) << endl;
13             cin >> a >> b >> c;
14         }
15     }
16     catch(double) {
17         cout << "Error: a=" << a << " "
18             << "b=" << b << " "
19             << "c=" << c << " "
20             << "it is not a triangle"
21             << endl;
22     }
23     cout << "end" << endl;
24     return 0;
25 }
27 double triangle(double x, double y, double z)
28 {
29     double s = (x+y+z)/2;
30     if(x+y<=z || y+z<=x || x+z<=y) throw x;
31     return sqrt(s*(s-x)*(s-y)*(s-z));
32 }
```

例14.2 异常处理2

其中，`GetNetworkResource()`通过网络获得数据，而两种异常类型都是从`std::exception`类派生的自定义类。需要注意的是推荐在`throw`时使用表示异常的对象，而在`catch`时以常引用形式捕捉异常（即对象）。

```
1  MyData md;  
2  try {  
3      // Code that could throw an exception  
4      md = GetNetworkResource();  
5  }  
6  catch (const networkIOException& e) {  
7      // Code that executes when an exception of type  
8      // networkIOException is thrown in the try block  
9      // ...  
10     // Log error message in the exception object  
11     cerr << e.what();  
12 }  
13 catch (const myDataFormatException& e) {  
14     // Code that handles another exception type  
15     // ...  
16     cerr << e.what();  
17 }  
18  
19 // The following syntax shows a throw expression  
20 MyData GetNetworkResource()  
21 {  
22     // ...  
23     if (IOSuccess == false)  
24         throw networkIOException("Unable to connect");  
25     // ...  
26     if (readError)  
27         throw myDataFormatException("Format error");  
28     // ...  
29 }
```

□ try-catch结构的说明：

1. 需要检测的代码必须放在try子句中，否则不起作用；
2. try子句和catch子句是一个完整的结构，catch子句必须紧跟try子句之后，不能独立使用，而不能在两子句之间插入其他语句；
3. try子句和catch子句必须使用复合语句的形式，即使只有一条语句；
4. 一个try-catch结构中只能有一个try子句，但可以有多多个catch子句；
5. catch子句中的异常信息通常是C++中的数据类型，例如catch(double)。catch只检查类型，而不检查具体的值，因此对于double a, b, c来说，throw a或throw b或throw c都会匹配catch(double)。

catch子句中的异常信息中还可以指定变量名，例如catch(double x)，此时如果throw a，则catch在捕获异常信息double的同时，还使用a的值给变量x进行赋值，这样可以获得关于异常的更多信息。

7. 如果在catch子句中没有指定异常信息的类型，而使用了删节符号“...”，则表示它可以捕捉任何类型的异常信息，例如 `catch(...){cout << "Error!" << endl;}`；
8. try-catch结构与throw既可以出现在同一函数中，也可以不在同一个函数中。当throw异常信息后，首先在本函数寻找匹配的catch子句，如果在本函数无try-catch结构或找不到匹配的catch，则转到其上一级函数去处理，如果还是无法处理，再向上转到更高一级的函数进行处理，以此类推；
9. 在某些情况下，在throw子句中可以有表达式，例如在catch子句中包含空throw子句，即

`catch(int){throw;}` 表示“我不处理这个异常，请上级处理”，catch子句把当前的异常信息再次throw，交给其上一级的catch子句进行处理；
10. 如果throw的异常信息找不到匹配的catch子句，那么系统就会调用一个系统函数 `terminate`，使程序终止运行。

□ throw的使用

- throw中抛出的异常信息推荐使用std::exception类或标准库中定义的类及它们的派生类。
如果没有合适的类，也可以从std::exception类中派生出自己的异常类。
- 虽然通常建议抛出std::exception类及其派生类，但 C++ 中其实可以抛出任何类型的异常。

throw的用法	含义
throw	将捕获的异常再次抛出
throw()	没有任何异常抛出
throw(...)	抛出任何一种异常，也可以没有异常
throw(type)	抛出type类型的异常

□ catch子句的使用

通过指定与抛出的异常具有相同类型或者可捕获任何类型的异常的 catch 子句对异常情况进行处理。

如果抛出的异常的类型是类，并还具有基类，则它可由捕获异常类型的基类和对异常类型的基类的引用的catch捕获。请注意，当捕获的是异常类型的引用，会将其绑定到实际抛出的异常对象；否则，它将是一个异常对象的副本。

◆ 抛出异常时，将由以下类型的 catch 子句捕获该异常：

- 可以捕获任何类型的catch子句（使用省略号语法），例如catch(...)；
- 捕获异常对象的类型的catch子句；
 - 由于它是副本，因此 const 和 volatile 修饰符将被忽略。
- 捕获异常对象的类型的引用的catch子句；

- 捕获异常对象的类型的 `const` 或 `volatile` 形式的引用的`catch`子句；
- 捕获异常对象的类型的基类的`catch`子句；
 - 由于它是副本，因此 `const` 和 `volatile` 修饰符将被忽略。
 - 基类的 `catch`子句不得位于派生类的 `catch`子句的前面。
- 捕获异常对象的类型的基类的引用的`catch`子句。
- 捕获异常对象的类型的基类的 `const` 或 `volatile` 形式的引用的`catch`子句。
- 捕获由数组名或函数名转换成的数组指针或函数指针的`catch`子句。

`catch`子句出现的顺序是有意义的，因为给定d `catch`子句按它们的出现顺序进行检查。例如，将基类的`catch`子句放置在派生类的`catch`子句的前面是错误的。找到一个匹配的 `catch`子句后，不会检查后续`catch`子句。因此，`catch(...)`必须是try-catch结构中的最后一个`catch`子句。

14.1.3 在函数声明中进行异常指定

异常指定是函数声明的一部分，必须同时出现在函数声明和函数定义的头部中，否则在进行该函数的另一次声明时，编译系统会报告“类型不匹配”。

例如：`double triangle(double, double, double) throw(double, int, char);`

表示triangle函数只限于抛出double、int或char类型的异常信息。

- **如果没有在函数声明中列出可能的异常类型，则该函数可以抛出任何类型的异常。**
- **如果声明一个不能抛出异常的函数，则应该声明形式如下：**

`double triangle(double, double, double) throw();`

这样，即使在函数执行过程中出了throw语句，但并不执行，程序将非正常终止。

14.1.4 在异常处理中处理析构函数

如果在try子句（或try子句中调用的函数）中定义了类对象，则在建立该对象时要调用构造函数。一旦发生异常，程序流程立即离开try子句（如果是在try子句调用的函数中发生异常，则流程先离开该函数，回到调用它的try子句中，然后从try子句中跳出转到catch子句）。这样流程就可能离开该类对象的作用域范围，因此C++的异常处理机制会在catch捕获异常时，对相关的局部对象进行析构，然后执行catch子句中的处理。

14.2 命名空间

命名空间，实际上是一个由程序设计者命名的内存区域。

程序设计者根据需要制定一些有名字的内存区域（即命名空间），把一些全局数据的名字分别放在各个命名空间中，从而实现与其他全局数据名的分隔，避免同名冲突。

在声明一个命名空间时，该命名空间可以包括以下类型：

- **变量（可以进行初始化）**
- **常量**
- **函数（可以是定义或声明）**
- **结构体**
- **类**
- **模板**
- **命名空间（即嵌套的命名空间）**

例如：

```
namespace ns1 {  
  
    const int RATE = 0.08;  
  
    double pay;  
  
    double tax() {return a*RATE}  
  
    namespace ns2 {int age;}  
  
}  
  
cout << ns1::RATE << ns1::pay << ns1::tax() << ns1::ns2::age << endl;
```

14.2.3 使用命名空间成员的方法

命名空间名::命名空间成员名

(1) 使用命名空间的别名

```
namespace Television { ... }  
namespace TV = Television;  
cout << TV::price << endl;
```

(2) 使用 “using 命名空间成员名”

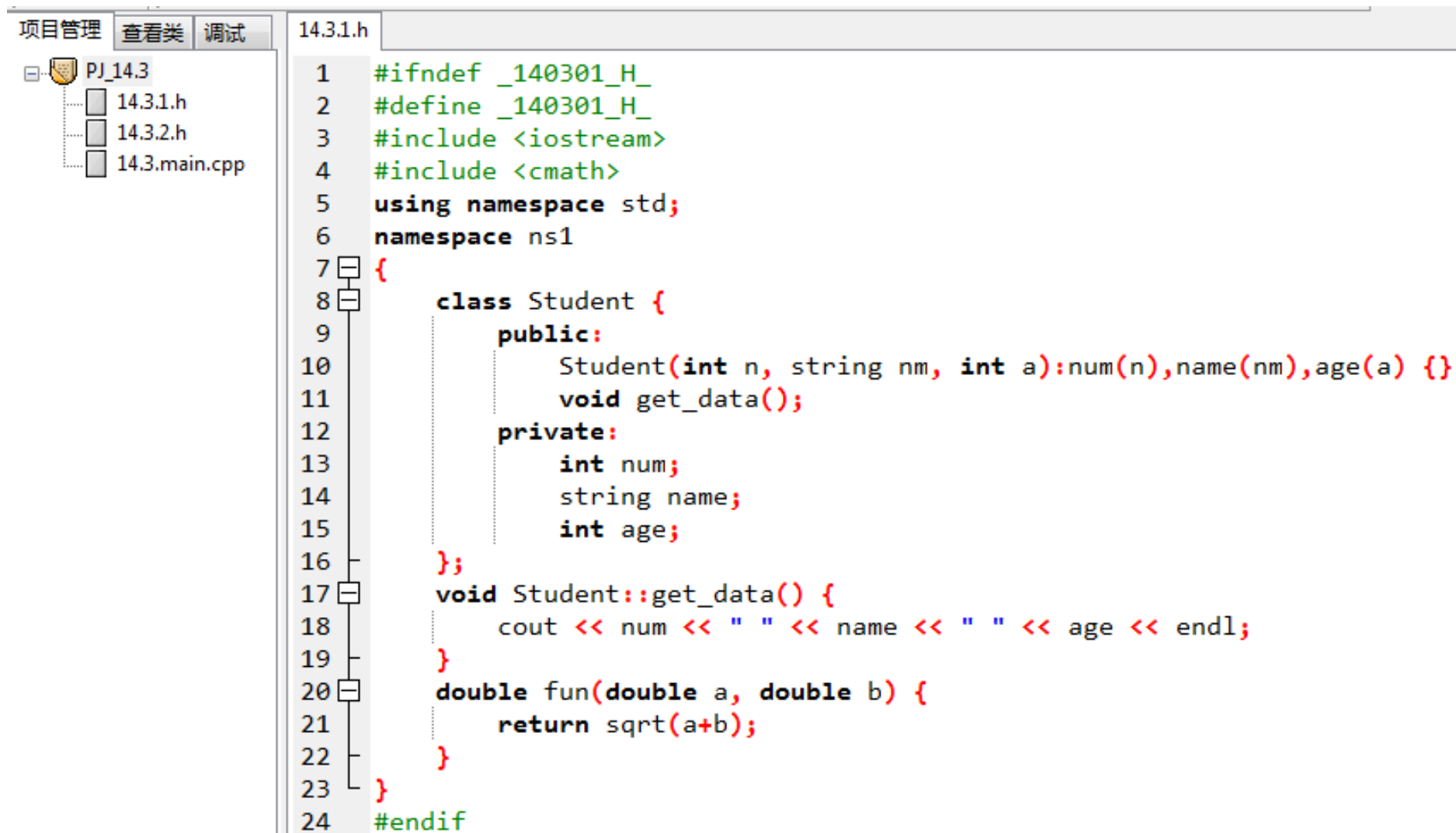
```
using ns1::tax; //在此声明的有效范围内可直接引用成员函数tax()  
cout << tax() << endl;
```

(3) 使用 “using namespace 命名空间名”

```
using namespace ns1; //此声明的有效范围内可直接引用该命名空间中的所有成员  
cout << RATE << pay << tax() << endl;
```

说明：在 (2) 中，如果使用using同时声明了不同命名空间中的同名成员；或在 (3) 中，如果使用using同时声明了不同命名空间，而这些命名空间中具有同名成员，则直接引用不同命名空间中的同名成员就产生二义性，造成编译出错。

例14.3 使用命名空间解决同名冲突



```
1  #ifndef _140301_H_
2  #define _140301_H_
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6  namespace ns1
7  {
8      class Student {
9      public:
10         Student(int n, string nm, int a):num(n),name(nm),age(a) {}
11         void get_data();
12     private:
13         int num;
14         string name;
15         int age;
16     };
17     void Student::get_data() {
18         cout << num << " " << name << " " << age << endl;
19     }
20     double fun(double a, double b) {
21         return sqrt(a+b);
22     }
23 }
24 #endif
```

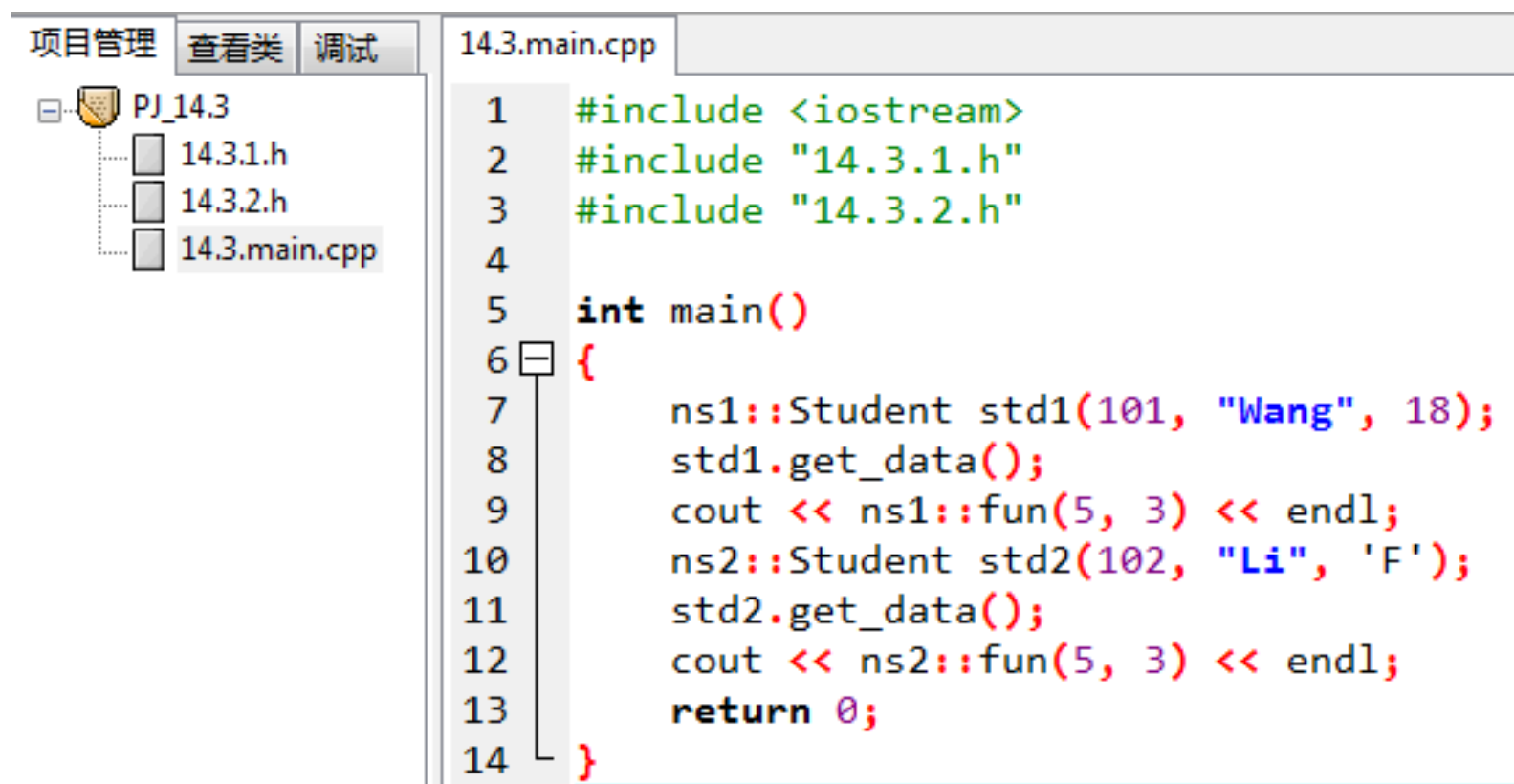
项目管理 查看类 调试

PJ_14.3

- 14.3.1.h
- 14.3.2.h
- 14.3.main.cpp

14.3.2.h

```
1  #ifndef _140302_H_
2  #define _140302_H_
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6  namespace ns2
7  {
8      class Student {
9      public:
10         Student(int n, string nm, char s):num(n),name(nm),sex(s) {}
11         void get_data();
12     private:
13         int num;
14         string name;
15         char sex;
16     };
17     void Student::get_data() {
18         cout << num << " " << name << " " << sex << endl;
19     }
20     double fun(double a, double b) {
21         return sqrt(a-b);
22     }
23 }
24 #endif
```



14.2.4 无名的命名空间

在文件A中声明无名命名空间

```
namespace { void fun() {cout << "OK" << endl;} }
```

由于该命名空间没有名字，因此只在本文件中有效，其他文件无法使用该命名空间。其成员函数fun的有效范围是从无名命名空间的声明位置开始到文件A的结尾。

14.2.5 标准命名空间std

标准C++库的所有标识符都是在一个名为std的命名空间中定义的，或者说标准头文件中的函数、类、对象和类模板都是在命名空间std中定义的。

14.3 使用C的标准函数库

C++保留了C的标准函数库，因此在C++中可以使用两种形式包含C的标准头文件。

(1) 直接使用C的标准头文件 (.h)

例如：#include <math.h>，此时不必声明标准命名空间std。

(2) 使用C++风格的C标准头文件

对C标准头文件名去掉.h后缀，并以字母c开头，就成为了C++风格的头文件名。

例如：#include <cmath>，此时需要声明标准命名空间std。

目前，大多数C++编译系统都同时支持以上两种用法，它们是等价的，可以任选其一。