

第五章 语句

简单语句

C++大多数语句以分号结束

表达式语句

含义：表达式末尾加上分号

特点：执行表达式并丢弃掉求值结果

如何理解丢掉求值结果？

```
1 ival + 5;    // 实际上这条表达式语句并没有作用，因为ival+5的结果被丢弃
2 cout<<val;    // 这条表达式实际上执行完也会丢弃结果，但是它本身执行就有
    附加的效果，就是将val的值绑定给cout标准输出流上显示在屏幕上
```

空语句

含义：只有单独的分号

常用情况：循环的全部工作在条件部分就可以完成时，为了循环语句的完整性，还是用于一个空语句来代表循环体内代码

- 使用空语句应该加上注释，从而令读代码的人知道该语句是有意省略
- 不要漏写分号，也别多写分号，多余的分号并非总是无害的（比如说循环语句）

复合语句（块）

含义：花括号括起来的（**可能为空**）语句和声明的序列

```
1 {}    //空块
```

空块的作用等价于空语句，当仍然被看作一条语句

特点：

- 如果一个程序的某个地方，语法上需要一条语句，但是逻辑上需要多条语句，则应该使用复合语句

复合语句可以看作一条语句，只是这条语句是一系列语句的组合

- 块不以分号作为结束

条件语句

if

switch

注意：

- **case标签** 必须是 **整型常量表达式**，且不能重复

整型 = 整数+字符型+布尔类型

- case匹配成功后，执行流程是顺序的，所以要加 **break** 语句

建议再最后一个语句后也加上 **break**，保持一致

- **default标签**，前面没有一个匹配就会跳到该标签执行语句

即使不准备在 **default** 标签下做任何工作，建议仍然使用。其目的在于告诉读者，我们已经考虑到了默认的情况，只是目前什么都不用做

**** switch 内部的变量定义****

- C++语言规定，不允许跨过 **变量的初始化语句** 直接跳转到该变量域内的另一个位置

```
1 case true:
2     string file_name;    // 编译错误，控制流绕过一个隐式初始化变量
3     int j =1;           // 编译错误，控制流绕过一个显式初始化变量
4     int i;              // 正确，因为i没有被初始化
5     break;
6 case false:
7     i = 1;              // 正确，即使int i;被跳过，它仍然有效
8     cout<<i;
```

- 在C++中，switch-case中的case实质上只是一个标签（label），**case中的代码并没有构成一个局部作用域**，虽然它的缩进好像是一个作用域

循环语句

while语句

- `while` 条件部分可以是一个表达式或者是一个带初始化的变量声明

for语句

传统for语句

- 顺序：
 1. 循环开始，先执行一次初始化语句；
 2. 判断条件语句，如果为真，执行循环体内语句
 3. 执行表达式语句
 4. 执行条件语句，进行判断

范围for语句

- 仅支持序列

序列：用于能返回迭代器的 `begin` 和 `end` 成员

STL容器是一种序列，但是序列不仅仅有STL容器

- 定义来源于与之等价的传统for语句

```
1 for(auto beg = v.begin(),end = v.end();beg!=end;++beg){  
2     ...  
3 }
```

所有范围for语句的对象必须包含 `begin(),end()` 对象；

并且在循环内部不能添加（删除）元素；

do-while语句

- 不允许在while括号内定义变量
- while结束后需要添加一个分号表示语句结束

跳转语句

**** break ****

跳出离它最近的循环

```
** conitnue **
```

终止最近的循环中的当前迭代并立即开始下一次迭代

```
** goto **
```

无条件从 `goto`语句 跳转到**同一函数**内的另一条语句

尽量不要使用 `goto`语句

```
1 // goto 语法
2 goto <label>; // 跳转到<label>标签处的语句继续指向
3
4 <label>: return
```

try语句块和异常处理

检测出问题的部分应该发出某种信号以表明程序遇到了故障，无法继续下去，而且信号的发出方无须知道故障将在何处得到解决。

如果程序中含有可能引发异常的代码，那么通常也会有专门的代码处理问题

- C++异常处理类别

- `throw`表达式：**异常检测部分**使用`throw`表达式来表示它遇到了无法处理的问题

`throw`引发了异常

- `try`语句块：**异常处理部分**使用`try`语句块处理异常。`try`语句块以关键字`try`开始，并以多个`catch`字句结束

`try`语句块中代码抛出的异常通常会被某个`catch`子句处理。

`catch`字句“处理”异常，所以它们也被称为异常处理代码

- 一套异常类：用于在`throw`表达式和相关的`catch`子句之间传递异常的具体信息
- 在真实的程序中，一般会把执行的代码和用户交互的代码分离开了。

throw表达式

```
1 // throw 语法
2 throw 异常类型;
```

- `throw`抛出一个异常，会终止**当前的函数**，并把控制权交给能处理该异常的代码

```

1 // try 语法
2 try{
3     可能会抛出异常的程序代码
4 }catch(异常声明){
5
6 }catch(...){
7
8 }

```

- 执行 try 块中的语句，如果执行的过程中没有异常抛出，那么执行完后就执行最后一个 catch 块后面的语句，**所有 catch 块中的语句都不会被执行**；
- 如果 try 块执行的过程中抛出了异常，那么抛出异常后**立即跳转到第一个“异常类型”和抛出的异常类型匹配的 catch 块中执行**（称作异常被该 catch 块“捕获”），执行完后再跳转到最后一个 catch 块后面继续执行。

函数在寻找处理代码的过程：（try语句嵌套情况）

如果异常被抛出，首先搜索抛出该异常的函数。如果没有找到匹配的catch语句，终止该函数，并在调用的函数中继续寻找。

如果还没找到，继续下去。

如果最终还是没能找到，程序转到名为 `terminate` 的标准库函数，使得程序非正常退出

catch语句中，最好使用引用类型的异常

标准异常

C++ 标准库定义了一组类，用于报告标准库函数遇到的问题，它们分别定义在4个头文件中：

- `<exception>`头文件：定义了最通用的异常类。它只报告异常的发生，不提供任何格外的信息。
- `<stdexcept>`头文件：定义了几种常用异常类

| 异常类 | 含义 |
|------------------|-------------------------|
| exception | 最常见的问题 |
| runtime_error | 只有在运行时才能检测出来的问题 |
| range_error | 运行时错误：计算结果超出有意义的值域范围 |
| overflow_error | 运行时错误：计算上溢 |
| underflow_error | 运行时错误：计算下溢 |
| logic_error | 程序逻辑错误 |
| domain_error | 逻辑错误：参数对应的结果值不一样 |
| invalid_argument | 逻辑错误：无效参数 |
| length_error | 逻辑错误：试图创建一个超出该类型最大长度的对象 |
| out_of_range | 逻辑错误：使用一个超出有效范围的值 |

- `<new>`头文件：定义了bad_alloc异常类型。

12.1节

- `<type_info>`头文件：定义了bad_cast异常类型。

19.2节

- exception、bad_alloc、bad_cast只能以默认初始化的方式初始化，不允许为这些对象提供初始值
- 其他异常类型的行为恰好相反：应该用string对象或C风格字符串初始化这些类型的对象，但是不允许使用默认初始化的方式

异常类型值只定义了一个名为 `what` 的成员函数，该函数没有任何参数，返回值是一个指向C风格的字符串的 `const char*`。

该字符串的目的是提供一些关于异常的文本信息。

如果异常类型有一个字符串初始值，`what`返回该字符串；

对于其他无初始值的异常类型来说，`what`返回的类型有编译器决定。

C++中异常处理与if判断

if-else 方式的好处在于更贴近与逻辑思维，性能优于Exception。相对于Exception，其缺点是，不适合OOP，语义不明显，不易于错误跟踪或错误提示较少，并且类型比较单一。

exception方法的好处在于是业务逻辑和异常处理分离（代码相对清晰），try中处理业务，catch中处理异常情况。在API设计中，可以设计Exception Handler来处理异常，使得层次分明。同时，更好的OOP的封装和多态性。缺点在于性能相对差。但是对于大型项目的话用Exception的方式无疑是最佳的

个人理解：使用if else的时候，基本上你是应该知道有什么错误类型，才能用，否则你就没有判断依据；一个动作可能引发多种错误，错误是不确定的，那这个时候可以声明多种catch来处理

术语

异常声明：位于catch字句中的声明，指定了该catch字句能处理的异常类型

异常处理代码： == catch字句

范围for语句： 在一个序列中进行迭代的语句

terminate： 一个标准库函数；当异常没有被捕捉时调用，用来终止当前程序