C++异常处理入门，C++ try catch入门

开发程序是一项“烧脑”的工作，程序员不但要经过长期的知识学习和思维训练，还要做到一丝不苟，注意每一个细节和边界。即使这样，也不能防止程序出错。

> 专家指出，长期作息不规律 + 用脑过度的危害很大，可能会诱发神经衰弱、失眠等疾病。我就是受害者之一，曾被失眠困扰了好几年，不但入睡困难，还容易早醒。程序员要注意劳逸结合，多去健身房，多跑步，多打球，多陪女朋友旅游等，千万不要熬夜，以为深夜写代码效率高，这样会透支年轻的身体。

程序的错误大致可以分为三种，分别是语法错误、逻辑错误和运行时错误：
1) 语法错误在编译和链接阶段就能发现，只有 100% 符合语法规则的代码才能生成可执行程序。语法错误是最容易发现、最容易定位、最容易排除的错误，程序员最不需要担心的就是这种错误。

2) 逻辑错误是说我们编写的代码思路有问题，不能够达到最终的目标，这种错误可以通过调试来解决。

3) 运行时错误是指程序在运行期间发生的错误，例如除数为 0、内存分配失败、数组越界、文件不存在等。C++ 异常（Exception）机制就是为解决运行时错误而引入的。

运行时错误如果放任不管，系统就会执行默认的操作，终止程序运行，也就是我们常说的程序崩溃（Crash）。C++ 提供了异常（Exception）机制，让我们能够捕获运行时错误，给程序一次“起死回生”的机会，或者至少告诉用户发生了什么再终止程序。

【例1】一个发生运行时错误的程序：

```
01.  #include <iostream>
02.  #include <string>
03.  using namespace std;
04.
05.  int main(){
06.      string str = "http://c.biancheng.net";
07.      char ch1 = str[100];  //下标越界，ch1为垃圾值
08.      cout<<ch1<<endl;
09.      char ch2 = str.at(100);  //下标越界，抛出异常
10.      cout<<ch2<<endl;
11.      return 0;
12.  }
```

运行代码，在控制台输出 ch1 的值后程序崩溃。下面我们来分析一下原因。

at() 是 string 类的一个成员函数，它会根据下标来返回字符串的一个字符。与 [ ] 不同，at() 会检查下标是否越界，如果越界就抛出一个异常；而 [ ] 不做检查，不管下标是多少都会照常访问。

> 所谓抛出异常，就是报告一个运行时错误，程序员可以根据错误信息来进一步处理。

上面的代码中，下标 100 显然超出了字符串 str 的长度。由于第 6 行代码不会检查下标越界，虽然有逻辑错误，但是程序能够正常运行。而第 8 行代码则不同，at() 函数检测到下标越界会抛出一个异常，这个异常可以由程序员处理，但是我们在代码中并没有处理，所以系统只能执行默认的操作，也即终止程序执行。

捕获异常

我们可以借助 C++ 异常机制来捕获上面的异常，避免程序崩溃。捕获异常的语法为：

```
try{
    // 可能抛出异常的语句
}catch(exceptionType variable){
    // 处理异常的语句
}
```

try 和 catch 都是 C++ 中的关键字，后跟语句块，不能省略 { }。try 中包含可能会抛出异常的语句，一旦有异常抛出就会被后面的 catch 捕获。从 try 的意思可以看出，它只是“检测”语句块有没有异常，如果没有发生异常，它就“检测”不到。catch 是“抓住”的意思，用来捕获并处理 try 检测到的异常；如果 try 语句块没有检测到异常（没有异常抛出），那么就不会执行 catch 中的语句。

这就好比，catch 告诉 try：你去检测一下程序有没有错误，有错误的话就告诉我，我来处理，没有的话就不要理我！

catch 关键字后面的 exceptionType variable 指明了当前 catch 可以处理的异常类型，以及具体的出错信息。我们稍后再对异常类型展开讲解，当务之急是演示一下 try-catch 的用法，先让读者有一个整体上的认识。

【例2】修改上面的代码，加入捕获异常的语句：

```
01.  #include <iostream>
02.  #include <string>
03.  #include <exception>
04.  using namespace std;
05.
06.  int main(){
07.      string str = "http://c.biancheng.net";
```

```
08.
09.     try{
10.         char ch1 = str[100];
11.         cout<<ch1<<endl;
12.     }catch(exception e){
13.         cout<<"[1]out of bound!"<<endl;
14.     }
15.
16.     try{
17.         char ch2 = str.at(100);
18.         cout<<ch2<<endl;
19.     }catch(exception &e){  //exception类位于<exception>头文件中
20.         cout<<"[2]out of bound!"<<endl;
21.     }
22.
23.     return 0;
24. }
```

运行结果：
(
[2]out of bound!

可以看出，第一个 try 没有捕获到异常，输出了一个没有意义的字符（垃圾值）。因为 `[]` 不会检查下标越界，不会抛出异常，所以即使有错误，try 也检测不到。换句话说，发生异常时必须将异常明确地抛出，try 才能检测到；如果不抛出来，即使有异常 try 也检测不到。所谓抛出异常，就是明确地告诉程序发生了什么错误。

第二个 try 检测到了异常，并交给 catch 处理，执行 catch 中的语句。需要说明的是，异常一旦抛出，会立刻被 try 检测到，并且不会再执行异常点（异常发生位置）后面的语句。本例中抛出异常的位置是第 17 行的 at() 函数，它后面的 cout 语句就不会再被执行，所以看不到它的输出。

说得直接一点，检测到异常后程序的执行流会发生跳转，从异常点跳转到 catch 所在的位置，位于异常点之后的、并且在当前 try 块内的语句就都不会再执行了；即使 catch 语句成功地处理了错误，程序的执行流也不会再回退到异常点，所以这些语句永远都没有执行的机会了。本例中，第 18 行代码就是被跳过的代码。

执行完 catch 块所包含的代码后，程序会继续执行 catch 块后面的代码，就恢复了正常的执行流。

为了演示「不明确地抛出异常就检测不到异常」，大家不妨将第 10 行代码改为 `char ch1 = str[100000000];`，访问第 100 个字符可能不会发生异常，但是访问第 1 亿个字符肯定

会发生异常了，这个异常就是内存访问错误。运行更改后的程序，会发现第 10 行代码产生了异常，导致程序崩溃了，这说明 try-catch 并没有捕获到这个异常。

关于「如何抛出异常」，我们将在下节讲解，这里重点是让大家明白异常的处理流程：

```
抛出（Throw）--> 检测（Try） --> 捕获（Catch）
```

发生异常的位置

异常可以发生在当前的 try 块中，也可以发生在 try 块所调用的某个函数中，或者是所调用的函数又调用了另外的一个函数，这个另外的函数中发生了异常。这些异常，都可以被 try 检测到。

1) 下面的例子演示了 try 块中直接发生的异常：

```
#include <iostream>
#include <string>
#include <exception>
using namespace std;

int main(){
    try{
        throw "Unknown Exception";  //抛出异常
        cout<<"This statement will not be executed."<<endl;
    }catch(const char* &e){
        cout<<e<<endl;
    }

    return 0;
}
```

运行结果：
Unknown Exception

`throw` 关键字用来抛出一个异常，这个异常会被 try 检测到，进而被 catch 捕获。关于 throw 的用法，我们将在下节深入讲解，这里大家只需要知道，在 try 块中直接抛出的异常会被 try 检测到。

2) 下面的例子演示了 try 块中调用的某个函数中发生了异常：

```
#include <iostream>
#include <string>
#include <exception>
using namespace std;
```

```
05.
06.  void func(){
07.      throw "Unknown Exception";  //抛出异常
08.      cout<<"[1]This statement will not be executed."<<endl;
09.  }
10.
11.  int main(){
12.      try{
13.          func();
14.          cout<<"[2]This statement will not be executed."<<endl;
15.      }catch(const char* &e){
16.          cout<<e<<endl;
17.      }
18.
19.      return 0;
20.  }
```

运行结果：
Unknown Exception

func() 在 try 块中被调用，它抛出的异常会被 try 检测到，进而被 catch 捕获。从运行结果可以看出，func() 中的 cout 和 try 中的 cout 都没有被执行。

3) try 块中调用了某个函数，该函数又调用了另外的一个函数，这个另外的函数抛出了异常：

```
01.  #include <iostream>
02.  #include <string>
03.  #include <exception>
04.  using namespace std;
05.
06.  void func_inner(){
07.      throw "Unknown Exception";  //抛出异常
08.      cout<<"[1]This statement will not be executed."<<endl;
09.  }
10.
11.  void func_outer(){
12.      func_inner();
13.      cout<<"[2]This statement will not be executed."<<endl;
14.  }
15.
16.  int main(){
17.      try{
18.          func_outer();
```

```
19.          cout<<"[3]This statement will not be executed."<<endl;
20.      }catch(const char* &e){
21.          cout<<e<<endl;
22.      }
23.
24.      return 0;
25.  }
```

运行结果：
Unknown Exception

发生异常后，程序的执行流会沿着函数的调用链往前回退，直到遇见 try 才停止。在这个回退过程中，调用链中剩下的代码（所有函数中未被执行的代码）都会被跳过，没有执行的机会了。