C++异常类型以及多级catch匹配

首先来回顾一下上节讲到的 try-catch 的用法：

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

我们还遗留下一个问题，就是 catch 关键字后边的 `exceptionType variable`，这节就来详细分析一下。

`exceptionType` 是异常类型，它指明了当前的 catch 可以处理什么类型的异常；`variable` 是一个变量，用来接收异常信息。当程序抛出异常时，会创建一份数据，这份数据包含了错误信息，程序员可以根据这些信息来判断到底出了什么问题，接下来怎么处理。

异常既然是一份数据，那么就应该有数据类型。C++ 规定，异常类型可以是 int、char、float、bool 等基本类型，也可以是指针、数组、字符串、结构体、类等聚合类型。C++ 语言本身以及标准库中的函数抛出的异常，都是 exception 类或其子类的异常。也就是说，抛出异常时，会创建一个 exception 类或其子类的对象。

`exceptionType variable` 和函数的形参非常类似，当异常发生后，会将异常数据传递给 variable 这个变量，这和函数传参的过程类似。当然，只有跟 exceptionType 类型匹配的异常数据才会被传递给 variable，否则 catch 不会接收这份异常数据，也不会执行 catch 块中的语句。换句话说，catch 不会处理当前的异常。

我们可以将 catch 看做一个没有返回值的函数，当异常发生后 catch 会被调用，并且会接收实参（异常数据）。

但是 catch 和真正的函数调用又有区别：

- 真正的函数调用，形参和实参的类型必须要匹配，或者可以自动转换，否则在编译阶段就报错了。
- 而对于 catch，异常是在运行阶段产生的，它可以是任何类型，没法提前预测，所以不能在编译阶段判断类型是否正确，只能等到程序运行后，真的抛出异常了，再将异常类型和 catch 能处理的类型进行匹配，匹配成功的话就“调用”当前的 catch，否则就忽略当前的 catch。

总起来说，catch 和真正的函数调用相比，多了一个「在运行阶段将实参和形参匹配」的过程。

另外需要注意的是，如果不希望 catch 处理异常数据，也可以将 variable 省略掉，也即写作：

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

这样只会将异常类型和 catch 所能处理的类型进行匹配，不会传递异常数据了。

多级 catch

前面的例子中，一个 try 对应一个 catch，这只是最简单的形式。其实，一个 try 后面可以跟多个 catch：

```
01.  try{
02.      //可能抛出异常的语句
03.  }catch (exception_type_1 e){
04.      //处理异常的语句
05.  }catch (exception_type_2 e){
06.      //处理异常的语句
07.  }
08.  //其他的catch
09.  catch (exception_type_n e){
10.      //处理异常的语句
11.  }
```

当异常发生时，程序会按照从上到下的顺序，将异常类型和 catch 所能接收的类型逐个匹配。一旦找到类型匹配的 catch 就停止检索，并将异常交给当前的 catch 处理（其他的 catch 不会被执行）。如果最终也没有找到匹配的 catch，就只能交给系统处理，终止程序的运行。

下面的例子演示了多级 catch 的使用：

```
01.  #include <iostream>
02.  #include <string>
03.  using namespace std;
04.
05.  class Base{ };
06.  class Derived: public Base{ };
07.
08.  int main(){
09.      try{
10.          throw Derived();  //抛出自己的异常类型，实际上是创建一个Derived类型的匿名对象
11.          cout<<"This statement will not be executed."<<endl;
```

```
    }catch(int){
        cout<<"Exception type: int"<<endl;
    }catch(char *){
        cout<<"Exception type: cahr *"<<endl;
    }catch(Base){  //匹配成功（向上转型）
        cout<<"Exception type: Base"<<endl;
    }catch(Derived){
        cout<<"Exception type: Derived"<<endl;
    }

    return 0;
}
```

运行结果：
Exception type: Base

> 在 catch 中，我们只给出了异常类型，没有给出接收异常信息的变量。

本例中，我们定义了一个基类 Base，又从 Base 派生类出了 Derived。抛出异常时，我们创建了一个 Derived 类的匿名对象，也就是说，异常的类型是 Derived。

我们期望的是，异常被 `catch(Derived)` 捕获，但是从输出结果可以看出，异常提前被 `catch(Base)` 捕获了，这说明 catch 在匹配异常类型时发生了向上转型（Upcasting）。

catch 在匹配过程中的类型转换

C/C++ 中存在多种多样的类型转换，以普通函数（非模板函数）为例，发生函数调用时，如果实参和形参的类型不是严格匹配，那么会将实参的类型进行适当的转换，以适应形参的类型，这些转换包括：

- 算数转换：例如 int 转换为 float，char 转换为 int，double 转换为 int 等。
- 向上转型：也就是派生类向基类的转换，请猛击《C++向上转型（将派生类赋值给基类）》了解详情。
- const 转换：也即将非 const 类型转换为 const 类型，例如将 char * 转换为 const char *。
- 数组或函数指针转换：如果函数形参不是引用类型，那么数组名会转换为数组指针，函数名也会转换为函数指针。
- 用户自定的类型转换。

catch 在匹配异常类型的过程中，也会进行类型转换，但是这种转换受到了更多的限制，仅能进行「向上转型」、「const 转换」和「数组或函数指针转换」，其他的都不能应用于 catch。

向上转型在上面的例子中已经发生了，下面的例子演示了 const 转换以及数组和指针的转换：

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

int main(){
    int nums[] = {1, 2, 3};
    try{
        throw nums;
        cout<<"This statement will not be executed."<<endl;
    }catch(const int *){
        cout<<"Exception type: const int *"<<endl;
    }

    return 0;
}
```

运行结果：
Exception type: const int *

nums 本来的类型是 `int [3]`，但是 catch 中没有严格匹配的类型，所以先转换为 `int *`，再转换为 `const int *`。

> 数组也是一种类型，数组并不等价于指针，这点已在《数组和指针绝不等价，数组是另外一种类型》和《数组到底在什么时候会转换为指针》中进行了详细讲解。