异常处理

- 在程序运行环境正常的情况下，程序不会出错。
- 但是一些特殊情况下，程序不能正确运行
  - 例：数组越界，输入的数据个数大于数组的长度
  - 例：除0错误，除数为0，造成程序溢出。
- 程序的鲁棒性（Robustness）
  - 程序在各种极端的情况下能够正确运行的程度
  - 为了提高鲁棒性，需要对程序的异常情况进行考虑并给出相应的处理。

异常概述

程序的错误通常包括：

- 语法错误。指程序的书写不符合语言的语法规则，这类错误可由编译程序发现。
- 逻辑错误（或语义错误）。指程序设计不当造成程序没有完成预期的功能，这类错误通过测试发现。
- 运行异常。指由程序运行环境问题造成的程序异常终止，如：内存空间不足、打开不存在的文件进行读操作、程序执行了除以0的指令等等。

- 例如：

```
void f(char *filename)
{ ifstream file(filename);
  int x;
  file >> x; //如果filename指定的文件不
             //存在，将会出现程序异常!
  ......
}
```

- 在程序运行环境正常的情况下，导致运行异常的错误是不会出现的。
- 程序异常错误往往是由于程序设计者对程序运行环境的一些特殊情况考虑不足所造成的。
  - 例如，除0错误，读不存在的文件
- 导致程序运行异常的情况是可以预料的，但它是无法避免的。
- 为了保证程序的鲁棒性，必须在程序中对可能的异常进行预见性处理。

处理异常的策略

- 就地处理（在发现错误的地方处理异常）
  - 调用C++标准库中的函数exit或abort终止程序执行。
  - 根据不同的情况给出不同的处理。
- 异地处理（在非异常发现地处理异常）
  - 通过函数的返回值或指针/引用类型的参数把异常情况通知函数的调用者。
  - 通过语言提供的结构化异常处理机制进行处理。

C++异常处理机制

C++异常处理机制的主要思想是：

- 把有可能造成异常的一系列操作（语句或函数调用）构成一个try语句块。
- 如果try语句块中的某个操作在执行中发现了异常，则通过执行一个throw语句抛掷（产生）一个异常对象。
- 抛掷的异常对象将由能够处理这个异常的地方通过catch语句块来捕获并处理之。

try语句

- try语句块的作用是启动异常处理机制。其格式为：

```
try
{ <语句序列>
}
```

- 上述的<语句序列>中可以有函数调用。例如，在调用函数f的函数中，可把对f的调用放在一个try语句块中：

```
try
{ f(filename);
}
```

throw语句

- throw语句用于在发现异常情况时抛掷（产生）异常对象。

  throw <表达式>;

  - <表达式>为任意类型的C++表达式（void除外）。

例如:

```
void f(char *filename)
{ ifstream file(filename);
  if (file.fail())
  { throw filename; //产生异常对象
  }
  ......
}
```

- 执行throw语句后，接在其后的语句将不再继续执行，而是转向异常处理（由某个catch语句给出）。

catch语句

- catch语句块用于捕获throw抛掷的异常对象并处理相应的异常。

```
catch (<类型> [<变量>])
{ <语句序列>
}
```

  - <类型>用于指出捕获何种异常对象，它与异常对象的类型匹配规则同函数重载；
  - <变量>用于存储异常对象，它可以缺省，缺省时表明catch语句块只关心异常对象的类型，而不考虑具体的异常对象。

- catch语句块要紧接在某个try语句的后面。

- 例如：

```
char filename[100];
cout << “请输入文件名：” << endl;
cin >> filename;
try
{ f(filename);
}
catch (char *str)
{ cout << str << “不存在!”<< endl;
  cout << “请重新输入文件名：” << endl;
  cin >> filename;
  f(filename);
}
```

- 如果在函数f中抛掷了char *类型的异常，则程序转到try后面的catch(char *str)处理。

- 一个try语句块的后面可以跟多个catch语句块，用于捕获不同类型的异常对象并进行处理。例如：

```
int main()
{ ......
  try
  { f();
  }
  catch (int) //处理函数f中的throw 1;
  { <语句序列1>
  }
  catch (double) //处理函数f中的throw 1.0
  { <语句序列2>
  }
  catch (char *) //处理函数f中的throw "abcd"
  { <语句序列3>
  }
  <非catch语句>
}
```

```
void f()
{ ......
  ...throw 1;
  ......
  ...throw 1.0;
  ......
  ...throw "abcd";
  .......
}
```

关于try、throw和catch几点注意

- 在try语句块的<语句序列>执行中如果没有抛掷（throw）异常对象，则其后的catch语句不执行，而是继续执行try语句块之后的非catch语句。

- 在try语句块的<语句序列>执行中如果抛掷（throw）了异常对象，并且该try语句块之后有能够捕获该异常对象的catch语句，则执行这个catch语句中的<语句序列>，然后继续执行这个catch语句之后的非catch语句。

- 在try语句块的<语句序列>执行中如果抛掷（throw）了异常对象，但其后没有能够捕获该异常对象的catch语句，则由函数调用链上的上一层函数中的try语句的catch来捕获。

- 如果抛掷异常对象的throw语句不是由程序中的某个try语句块中的<语句序列>调用的，则抛掷的异常不会被程序中的catch捕获。

异常处理的嵌套

try语句是可以嵌套的。当在内层的try语句的执行中产生了异常，则首先在内层try语句块之后的catch语句序列中查找与之匹配的处理，如果内层不存在能捕获相应异常的catch，则逐步向外层进行查找。

```
void f()
{ try
  { g();
  }
  catch (int)
  {     ......
  }
  catch (char *)
  { ......
  }
}
```

```
void g()
{ try
  { h();
  }
  catch (int)
  { ......
  }
}
```

```
void h()
{  ......
   ... throw 1; //由g捕获并处理
   ......
   ... throw "abcd"; //由f捕获并处理
   ......
}
```

例子：处理除数为0的异常

```
#include <iostream>
using namespace std;
int divide(int x, int y)
{ if (y==0) throw 0;
  return x/y;
 }
int main()
{ int a,b;
  while (true)
  {
     try
       {  cout <<“请输入两个数”;
          cin>>a>>b;
          int r=divide(a,b);
          cout << a<< “/” <<b << “=“ <<r << endl;
          break;
       }
      catch(int)
       {
         cout<<“除数不能为0，请重新输入”;
       }
   }
}
```

小结

- 异常的基本概念
- C++的异常处理机制
  - Throw
  - Try
  - Catch