火龙果•整理
uml.org.cn

C++ 异常处理

火龙果•整理
uml.org.cn

本章主要内容

- 异常处理的基本思想
- C++异常处理的实现
- 异常处理中的构造与析构

火龙果•整理
uml.org.cn

错误分类：

- 编译错

  编译时通不过，属语法错，最浅层次的错误。

- 运行错

  调试时无法发现，运行时才出现，往往由系统环境引起，属可予料但不可避免。必须由语言的某种机制予以控制。

- 逻辑错

  设计缺陷，编译器无法发现，只能靠人工分析跟踪排除，错误层次中等。

火龙果•整理
uml.org.cn

异常处理的指导思想

- 对于可予料但不可避免的程序错误，不能眼睁睁看着它发生而无所作为（只会用exit()），要将消极等待变为积极预防，还要将预防的处理内容归纳整理，分门别类地作成类。
- 积极之意是，不能只凭编程经验这种个体性偶然性的做法，而要凭借人人都掌握的规范圆满的处理。
- 这规范是指在函数的处理中设下陷阱，一旦触发异常，定会被异常处理所收容，统一归口处理。

火龙果•整理
uml.org.cn

异常处理的三种方式

- 在出现异常时，直接调用 abort() 或者 exit();
- 通过函数的返回值来判断异常。

  但是如果一个函数有多个返回值的时候会比较麻烦，而且也会浪费不必要的判断返回值的时间开销;
- 通过try{ } catch() 的结构化异常处理来完成;

火龙果•整理
uml.org.cn

异常处理的实现机制

呼应的三个关键字：try throw catch

- 抛掷异常的程序段

```
Func()
{
  throw  表达式;
}
```

- 捕获并处理异常的程序段

```
try
{
  复合语句
  func();
}
catch（异常类型声明）
  复合语句
catch（异常类型声明）
  复合语句
```

火龙果•整理
uml.org.cn

例12-1处理除零异常

```
int Div(int x,int y)
{   if(y==0) throw y;
    return x/y;
}

void main()
{   try
    { cout<<"5/2="<<Div(5,2)<<endl;
      cout<<"8/0="<<Div(8,0)<<endl;
      cout<<"7/1="<<Div(7,1)<<endl;
    }
   catch(int)
    { cout<<"except of deviding zero.\n"; }
    cout<<"that is ok.\n";
}
```

程序运行结果如下:

```
5/2=2
except of deviding zero.
that is ok.
```

火龙果•整理
uml.org.cn

异常处理的语法结构

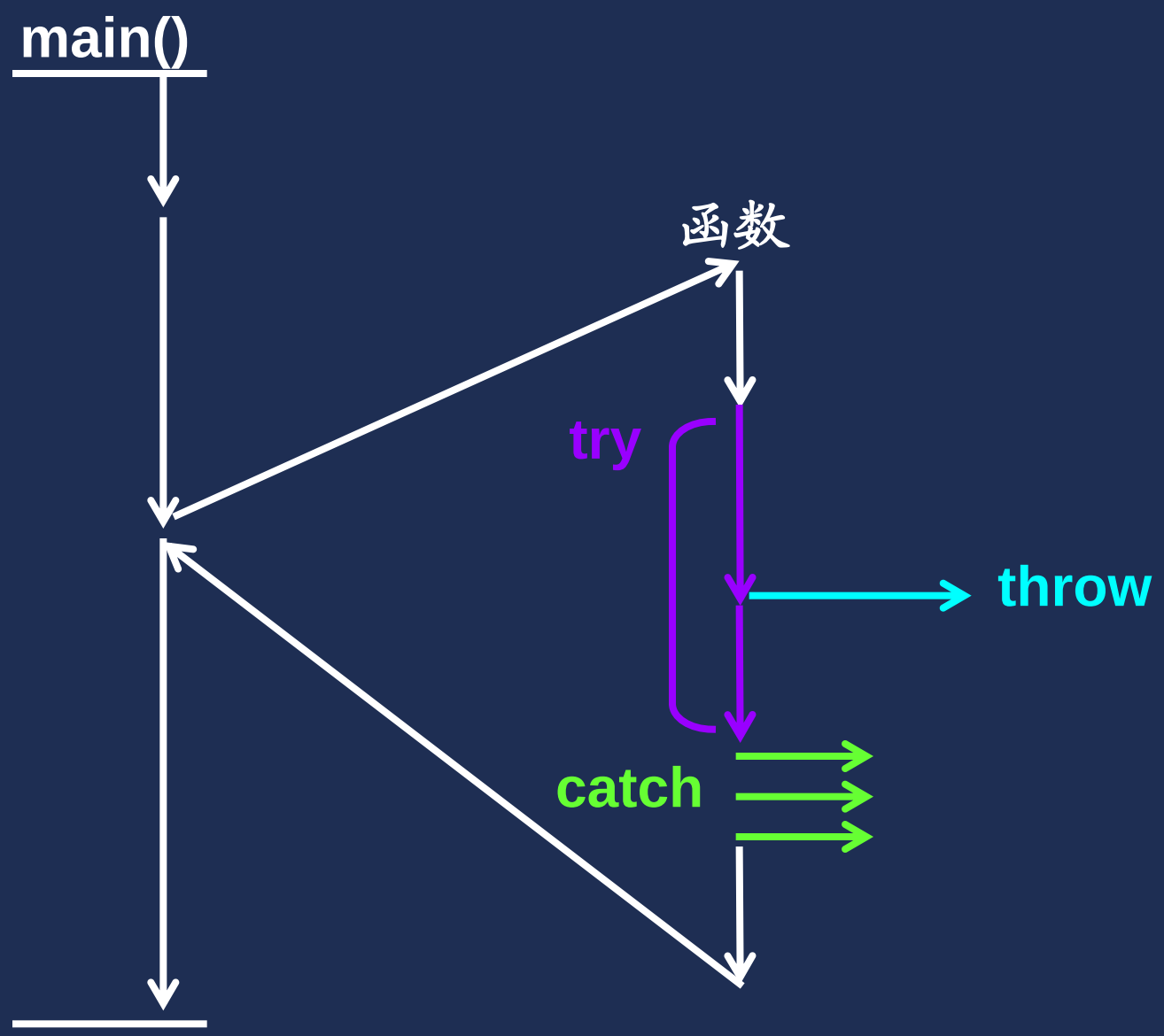
这三个关键字分工明确，各司其职:

- throw负责发现异常，抛掷异常对象；若放在函数声明中则又称为异常接口声明;
- try设置了一个侦错范围，又叫保护段。其实是划定了一个跳跃的边界;
- catch负责处理捕获来的异常（包括继续抛掷异常）。
- 所抛掷的异常对象并非建在函数栈上，而是建在专用的异常栈上，故可以跨越函数而传递给上层。

火龙果•整理
uml.org.cn

- 每个catch()相当于一段函数代码;
- 每个throw则相当于一个函数调用;
- 每个try块至少跟一个catch();
- 一个程序可设置个数不定的try 、throw 和 catch。它们只有逻辑上的呼应，而无数量上的对应关系，且不受所在函数模块限制;
- 异常抛掷点往往距异常捕获点很远，它们可以不在同层模块中;
- 甚至有的throw我们看不到在哪，实际上在我们所调用的系统函数中，在标准库中;
- 程序中try块可以并列、可以嵌套;

异常处理的模式



火龙果•整理
uml.org.cn

火龙果•整理
uml.org.cn

异常处理的不唤醒机制

- 即一旦在保护段执行期间发生异常，则立即抛掷，由相应的catch子句捕获处理。抛掷——捕获间的代码被越过而不执行。程序从try块后跟随的最后一个catch子句后面的语句继续执行下去。
- 异常处理是将检测与处理分离，以便各司其职、灵活搭配地工作。它们的联系靠类型。

火龙果•整理
uml.org.cn

异常处理的实现机制

- 若有异常则通过throw创建一个异常对象并抛掷。
- 将可能抛出异常的程序段嵌在try块之中。按正常的顺序执行到达try语句，然后执行try块内的保护段。
- 如果在保护段执行期间没有引起异常，那么跟在try块后的catch子句就不执行。程序从try块后跟随的最后一个catch子句后面的语句继续执行下去。
- catch子句按其在try块后出现的顺序被匹配。选中的catch将捕获并处理异常（或继续抛掷异常）。
- 如果匹配的处理器未找到，则运行函数terminate将被自动调用，其缺省功能是调用abort终止程序。

火龙果•整理
uml.org.cn

C++的异常处理机制有何特色？

C对出错的处理是将所有错误予以编码(返回码)，一码一错。程序员编程时，通过在主调函数中检查出现的返回码，给与相应的处理。这种方法繁琐又不规范。

C++则是将形形色色的错误归结抽象为“类型错”，统统交给throw抛掷；将出错处理统一为catch 调用。这就为程序员编程制定了统一的规范，也就是运行协议。

“异常处理”实际是个动态概念。其处理机制是靠类型匹配，这恰是运行时才能发生的事。

C对出错处理的返回码机制，与C++的异常处理机制有何不同？

火龙果•整理
uml.org.cn

1. 将正常逻辑与异常逻辑混在一起。每个函数调用后都要用正常逻辑（如if）加以判断，以过滤出异常。这样使子函数的错误扩散到了其所有的调用者中，形成了一个混杂的怪异链。
2. 增加了用于处理很少出现的特殊情况的运行费用。尤其对于C++，其成员函数通常只有几行代码，但函数很多，若每个函数都使用返回码技术，则臃肿不堪。
3. 类中还有构造、拷贝构造等无返回值的函数，它们出错处理的权力不应被剥夺。
4. 返回码靠的是预定的数字代号来传递信息，所能携带的信息量少且呆板，远少于抛掷的对象所携带的信息量。

例12-2 异常嵌套处理

火龙果•整理
uml.org.cn

```
int Div(int x,int y)
{
        if(y==0) throw y;
        return x/y;
}
int Cal(int x,int y)
{
        return Div(x,y);
}
void main()
{       try
        {
                cout<<"5/2="<<Cal(5,2)<<endl;
                cout<<"8/0="<<Cal(8,0)<<endl;
                cout<<"7/1="<<Cal(7,1)<<endl;
        }
        catch( int )        { cout<<"except of deviding zero.\n";  }

        cout<<"that is ok.\n";
}
```

火龙果•整理
uml.org.cn

异常嵌套处理的基本思想

火龙果•整理
uml.org.cn

异常处理中的构造与析构

- 找到一个匹配的catch异常处理后:
  - 初始化参数。
  - 将从对应的try块开始到异常被抛掷处之间构造的（且尚未析构的）所有对象进行析构。
  - 从最后一个catch处理之后开始恢复执行。

火龙果•整理
uml.org.cn

例12-3 异常处理时的析构

```
#include <iostream.h>
void MyFunc( void );
class Expt
{ public:
   Expt(){};
   ~Expt(){};
   const char *ShowReason() const
   {
        return "Expt类异常。";
   }
};
```

火龙果•整理
uml.org.cn

```
class Demo
{ public:
   Demo()
   {        cout<<"构造 Demo."<<endl;
   }
   ~Demo()
  {        cout<<"析构 Demo."<<endl;
  }
};
void MyFunc()
{ Demo D;
  cout<<"在MyFunc()中抛掷Expt类异常。"<<endl;
  throw Expt();
}
```

火龙果•整理
uml.org.cn

```
void main()
{ cout<<"在main函数中。"<<endl;
  try
  { cout<<"在try块中，调用MyFunc()。" <<endl;
    MyFunc();
  }
catch( Expt E )
  { cout<<"在catch异常处理程序中。"<<endl;
    cout<<"捕获到Expt类型异常：";
    cout<<E.ShowReason()<<endl;
  }
catch( char *str )
  { cout<<"捕获到其它的异常："<<str<<endl;}
  cout<<"回到main函数。从这里恢复执行。"<<endl;
}
```

火龙果•整理
uml.org.cn

程序运行时输出：

在main函数中。

在try块中，调用MyFunc()。

构造 Demo.

在MyFunc()中抛掷Expt类异常。

析构 Demo.

在catch异常处理程序中。

捕获到Expt类型异常：Expt类异常。

回到main函数。从这里恢复执行。

课堂练习： 读代码，分析结果。

```
void  f();
class  T
{ public:
    T()
    {
        cout<<“constructor”<<endl;
        try{
            throw  “exception  1”;
        }
        catch(char *)
        { cout<< “exception  1”<<endl; }
        throw “exception”;
    }
    ~T()
    {  cout <<“destructor”;  }
};
```

```
void main()
{
    cout<<“main  function”<<endl;
    try{
            f() ;
        }
        catch(char *)
         { cout<< “exception  2”<<endl; }
   cout<<“main  function”<<endl;
}
 void  f()
  {
        T  t;
  }
```

课堂练习： 读代码，分析结果 -- 答案

```
void  f();
class  T
{ public:
    T(){ cout<<"constructor"<<endl;          //  2
        try{
            throw  "exception  1";
        }
        catch(char *)
        { cout<< "exception  1"<<endl; }   //  3
        throw "exception";      //  将被执行
    }
    ~T()
    {  cout <<"destructor";  }   //  不执行
};
```

火龙果•整理
uml.org.cn

火龙果•整理
uml.org.cn

```
void main()
{
    cout<<"main  function"<<endl;    // 1
    try{
            f() ;
        }
        catch(char *)
         { cout<< "exception  2"<<endl; }      // 4
   cout<<"main  function"<<endl;              // 5
}
 void  f()
  {
       T  t;
  }
```

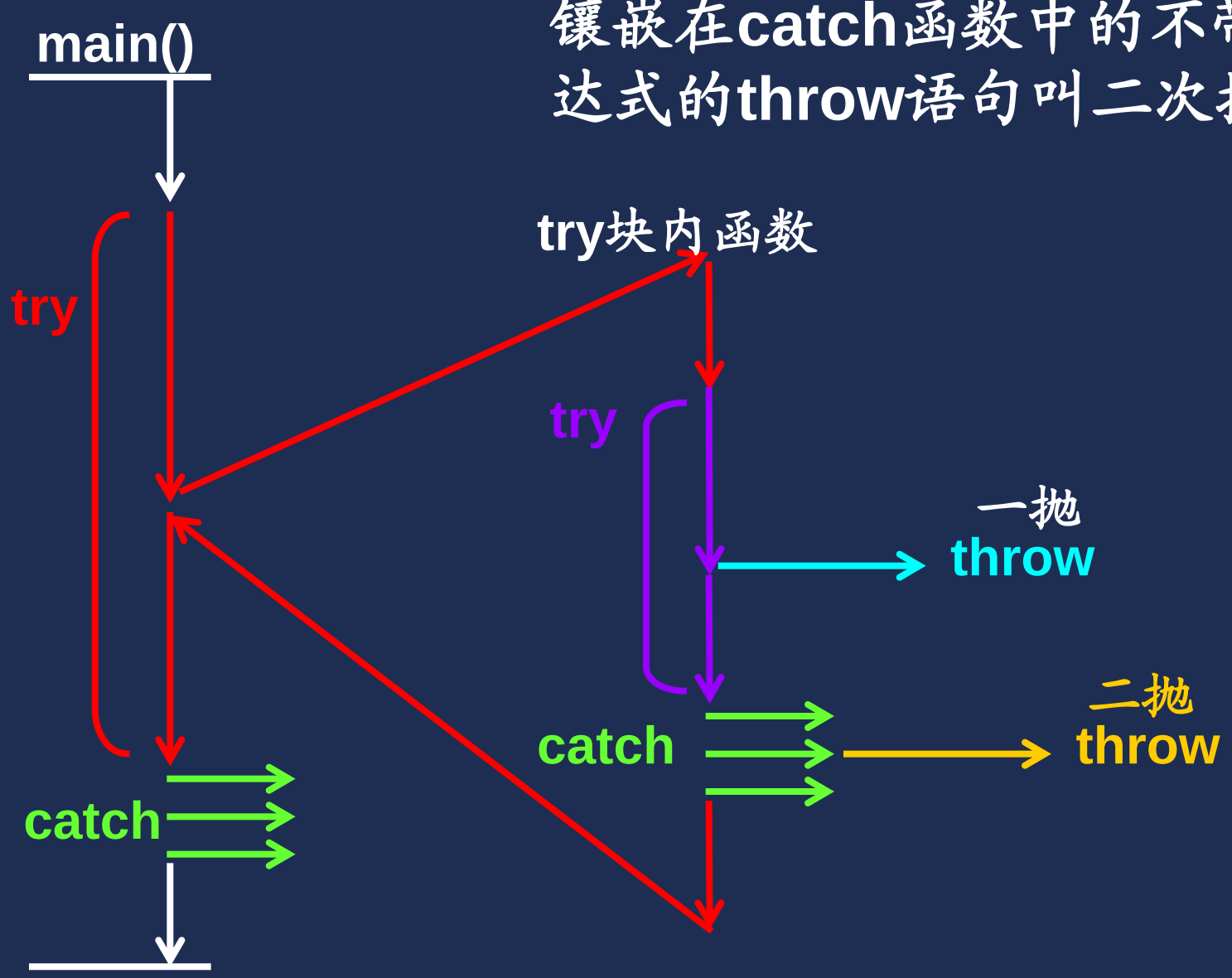
结论:

构造函数中含有异常处理，析构函数没有运行。

二次抛掷的图示

火龙果•整理
uml.org.cn

镶嵌在catch函数中的不带表达式的throw语句叫二次抛掷.



练习二：分析其中的调用及抛掷关系(将…改成

火龙果•整理
uml.org.cn

```
void f()   {....
   throw   10;
}
void   g()  {.....
   throw  3.2;
}
void   h()   {  ....
   throw  65.5;
   try{   ....
         throw  'a';
         f();    g();   }
  catch( float )
  { .....
         throw;
  }
}
```

```
void k()
{....
try{   ....
         h();     }
  catch( float )
  { .....
         throw   10;
  }
   catch( int )
  { .....
   }
}
void m()
{....
   k();
}
```

```
void main()
{   ....
try
{   ....
         m();
}
  catch( int )
  { .....     }
   catch( float )
  { .....     }
  catch( ... )
  { .....
   }
}
```

如果把float换成double呢？

火龙果•整理
uml.org.cn

Windows 异常处理

火龙果•整理
uml.org.cn

Windows对于C++异常处理的扩展：

- 为了更便捷的在Windows系统中进行异常处理，微软在C++异常处理的机制上，又增加了扩展，称为SEH（Structured Exception Handling），即结构化异常处理；
- 语法如下：

```
__try
{
    ......
}
__except (<exception>)
{
    ......
}
```

__try - __except：

火龙果•整理
uml.org.cn

- __try-__except是Microsoft扩展出的C++关键字，__try块中出现错误或异常，一般不再用throw抛出，而是直接产生一个EXCEPTION_POINTERS类型的异常数据，然后开始查找SEH例程入口（调试的情况除外）。首先就会找到与__try块对应的__except块。__except的参数<exception>与catch的参数作用完全不同，也不类似于函数的参数，它主要是用于控制后面的程序执行，为这几个值之一：
- EXCEPTION_EXECUTE_HANDLER（1），表示下面执行__except块内及其后面的代码
  EXCEPTION_CONTINUE_EXECUTION（-1），表示回到抛出异常处继续向下执行
  EXCEPTION_CONTINUE_SEARCH（0）, 表示查找下一个异常处理例程入口
- Microsoft提供两个函数GetExceptionCode(), GetExceptionInformation()，分别可以获取异常号和EXCEPTION_POINTERS类型的异常数据指针。而且这两个函数只能在__except参数<exception>的表达式中使用。为了保证这一点，在VC中，编译器做了特殊处理，如果这两个函数没有在正确的位置，将产生编译错误。（这个很搞笑。）

例： 处理除零异常：

火龙果•整理
uml.org.cn

```
Int main()          #include <windows.h>
{
        int i = 1;
        int j = 0;

        __try
        {
                i /= j;
        }
        __except(GetExceptionCode() ==
                                EXCEPTION_INT_DIVIDE_BY_ZERO ?
                                EXCEPTION_EXECUTE_HANDLER :
                                EXCEPTION_CONTINUE_SEARCH)
        {
                cout << “除零异常" << endl;
        }
}
```

例：处理除零异常(简写)

火龙果•整理
uml.org.cn

```
Int main()
{
        int i = 1;
        int j = 0;

        __try
        {
                i /= j;
        }
        __except(  EXCEPTION_EXECUTE_HANDLER )
        {
                cout << “除零异常" << endl;
        }
}
```

例：处理写内存异常：

火龙果•整理
uml.org.cn

```
Int main()
{
        __try
        {
                int *p=NULL;
                *p = 13;
        }
        __except(  EXCEPTION_EXECUTE_HANDLER )
        {
                cout << “内存异常" << endl;
        }
}
```

火龙果•整理
uml.org.cn

Windows异常处理流程：

- （1）中止当前程序的执行。
- （2）如果程序处于被调试状态，向调试器发送EXCEPTION_DEBUG_EVENT消息。
- （3）如果程序没有被调试或者调试器未能处理异常，查找线程相关的异常处理例程（如对应__except块）并处理。如果前面查找到的例程返回EXCEPTION_CONTINUE_SEARCH，且线程有多个异常处理例程，则沿这些例程入口地址组成的链式结构逐一向后查找，请求下一个例程处理。
- （4）如果线程没有对应的异常处理例程，或线程所有例程都返回EXCEPTION_CONTINUE_SEARCH，而且程序处于被调试状态，再次通知调试器。
- （5）如果程序没有被调试或者调试器仍未处理异常，则进入主线程的“最终异常处理例程”链继续查找。
- （6）“最终异常处理例程”链的最后是Windows默认的系统异常处理程序__CxxUnhandledExceptionFilter()，其处理通常是弹出一个异常对话框，上面显示一些异常信息，提供“关闭”、“调试”等按钮。

火龙果•整理
uml.org.cn

C++和Windows异常处理比较：

	C++标准异常	SEH
局部对象析构函数	执行	局部对象有析构函数，且用__try-__exception时，编译错误
可移植	C++都有，不依赖操作系统平台	只有Windows提供
程序数据通用 流程控制	catch块后只能继续向下执行	EXCEPTION_EXECUTE_HANDLER、EXCEPTION_CONTINUE_EXECUTION、EXCEPTION_CONTINUE_SEARCH三种流程控制，多个处理例程的依次处理
数据通用	各种不同的异常数据类型	统一结构的异常数据