

知识点1【异常的概述】（了解）

知识点2【异常的抛出、捕获】（了解）

1、异常的抛出、捕获

知识点3【栈解旋】（了解）

知识点4【异常的接口声明】（了解）

知识点5【异常变量的生命周期】（了解）

1、以普通对象 接异常值

2、以对象指针 接异常值

3、对象引用 接异常值（推荐）

知识点6【异常的多态】（了解）

知识点6【c++标准异常】（重要）

知识点7【编写自己的异常】（了解）

知识点1【异常的概述】（了解）

遇到错误 **抛出**异常 **捕获**异常

异常：是指在程序运行的过程中发生的一些异常事件（如：**除0**溢出，**数组下标**越界，所要读取的文件不存在,空指针，内存不足，访问非法内存等等）。（异常是一个**类**）

c++异常机制相比C语言异常处理的优势？

函数的返回值可以忽略，但异常不可忽略。（忽略异常 程序结束）

整型返回值没有任何语义信息。而异常却包含语义信息，有时你从类名就能够体现出来

知识点2【异常的抛出、捕获】（了解）

1、异常的抛出、捕获

```
1  try
2  {
3      throw 异常值;
```

```

4 }
5 catch(异常类型1 异常值1)
6 {
7     处理异常的代码1;
8 }
9 catch(异常类型2 异常值2)
10 {
11     处理异常的代码2;
12 }
13 catch(...)//任何异常都捕获
14 {
15     处理异常的代码3;
16 }

```

案例1:

```

1 int ret = 0;
2 try
3 {
4     //throw 1;
5     //throw 'A';
6     throw 2.14f;
7
8 }
9 catch(int e)//捕获
10 {
11     cout<<"int异常值为:"<<e<<endl;
12 }
13 catch(char e)//捕获
14 {
15     cout<<"char异常值为:"<<e<<endl;
16 }
17 catch(...)//捕获所有异常
18 {
19     cout<<"其他异常值为:"<<endl;
20 }

```

其他异常值为:

知识点3 【栈解旋】（了解）

异常被抛出后，从进入try块起，到异常被抛掷前，这期间在栈上构造的所有对象，都会被自动析构。析构的顺序与构造的顺序相反，这一过程称为**栈的解旋**。

```
1  try
2  {
3      Data ob1;
4      Data ob2;
5      Data ob3;
6      throw 1; //抛出异常后 ob3 ob2 ob1依次自动释放（栈解旋）
7  }
```

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Data
6  {
7  public:
8      int a;
9  public:
10     Data(){}
11     Data(int a)
12     {
13         this->a = a;
14         cout<<"构造函数"<<a<<endl;
15     }
16     ~Data()
17     {
18         cout<<"析构函数"<<a<<endl;
19     }
20 };
21
22 void test01()
23 {
24     int ret = 0;
25     try
26     {
27         Data ob1(10);
28         Data ob2(20);
29         Data ob3(30);
```

```

30  throw 1;
31  }
32
33  catch(int)//捕获
34  {
35      cout<<"int异常值为:"<<endl;
36  }
37  catch(char)//捕获
38  {
39      cout<<"char异常值为:"<<endl;
40  }
41  catch(...)//捕获
42  {
43      cout<<"其他异常值为:"<<endl;
44  }
45  }
46
47  int main(int argc, char *argv[])
48  {
49      test01();
50      return 0;
51  }

```

```

构造函数10
构造函数20
构造函数30
析构函数30
析构函数20
析构函数10
int异常值为:

```

知识点4【异常的接口声明】（了解）

异常的接口声明：描述的是 可以抛出哪些类型的异常

```

1  #include <iostream>

```

```
2
3 using namespace std;
4
5 //函数默认 可以抛出任何类型的异常（推荐）
6 void fun01()
7 {
8     //throw 1;
9     //throw '1';
10    throw "hello";
11
12 }
13 //只能抛出int,char异常
14 void fun02() throw(int,char)
15 {
16     //throw 1;
17     //throw '1';
18     throw "hello";//抛出 不能捕获
19 }
20
21 //不能抛出 任何异常
22 void fun03() throw()
23 {
24     throw 1;
25     //throw '1';
26     //throw "hello";//抛出 不能捕获
27 }
28
29 void test01()
30 {
31     int ret = 0;
32     try
33     {
34         fun03();
35     }
36     catch(int)//捕获
37     {
38         cout<<"int异常值为:"<<endl;
39     }
40     catch(char)//捕获
41     {
```

```

42  cout<<"char异常值为:"<<endl;
43  }
44  catch(const char *)//捕获
45  {
46  cout<<"const char *异常值为:"<<endl;
47  }
48
49  catch(...)//捕获
50  {
51  cout<<"其他异常值为:"<<endl;
52  }
53  }
54
55  int main(int argc, char *argv[])
56  {
57  test01();
58  return 0;
59  }

```

知识点5 【异常变量的生命周期】（了解）

```

1  class MyException
2  {
3  public:
4  MyException(){
5  cout << "异常变量构造" << endl;
6  };
7  MyException(const MyException & e)
8  {
9  cout << "拷贝构造" << endl;
10 }
11 ~MyException()
12 {
13 cout << "异常变量析构" << endl;
14 }
15 };

```

1、以普通对象 接异常值

```

void test02()
{
    try
    {
        throw MyException();
    }
    catch(MyException e)//普通对象接异常(拷贝构造发生)
    {
        cout<<"普通对象接异常"<<endl;
    }
}

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator

异常变量构造
拷贝构造
普通对象接异常
异常变量析构
异常变量析构

2、以对象指针 接异常值

```

try
{
    throw new MyException;
}
catch(MyException *e)//指针对象接异常(delete释放)
{
    cout<<"普通对象接异常"<<endl;
    delete e;
}

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qt

异常变量构造
普通对象接异常
异常变量析构

3、对象引用 接异常值 (推荐)

```

try
{
    throw MyException();
}
catch(MyException &e)
{
    cout<<"普通对象接异常"<<endl;
}

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_

异常变量构造
普通对象接异常
异常变量析构

知识点6 【异常的多态】 (了解)

```

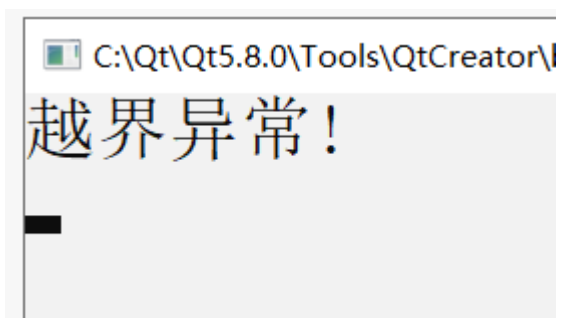
1 //异常基类
2 class BaseException{
3 public:
4     virtual void printError(){};
5 };
6
7 //空指针异常
8 class NullPointerException : public BaseException{
9 public:
10     virtual void printError(){
11         cout << "空指针异常!" << endl;

```

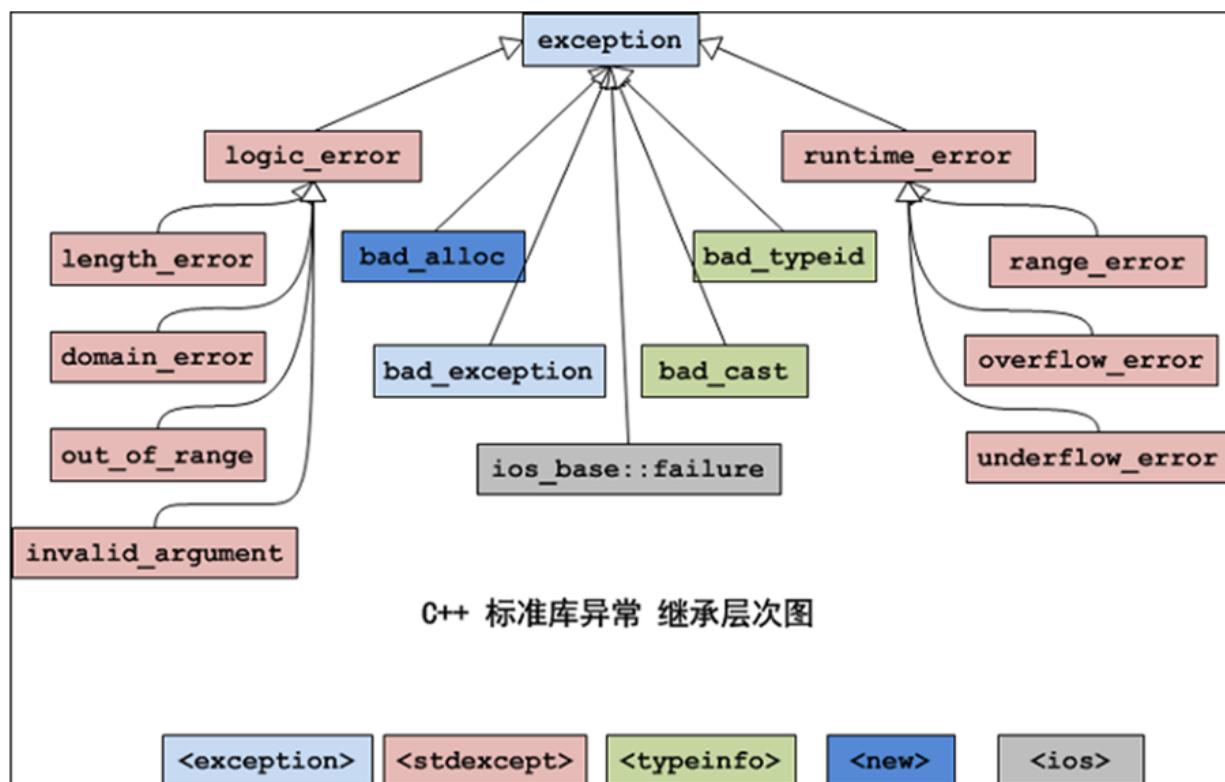
```

12  }
13  };
14  //越界异常
15  class OutOfRangeException : public BaseException{
16  public:
17      virtual void printError(){
18          cout << "越界异常!" << endl;
19      }
20  };
21
22  void doWork(){
23
24      //throw NullPointerException();
25      throw OutOfRangeException();
26  }
27
28  void test03()
29  {
30      try{
31          doWork();
32      }
33      catch (BaseException& ex)//父类引用 可以捕获搭配该父类派生出的所有子类的子类
34      {
35          ex.printError();
36      }
37  }

```



知识点6 【c++标准异常】（重要）



异常名称	描述
exception	所有标准异常类的父类
bad_alloc	当operator new and operator new[], 请求分配内存失败时
bad_exception	这是个特殊的异常，如果函数的异常抛出列表里声明了bad_exception异常，当函数内部抛出了异常抛出列表中没有的异常，这是调用的unexpected函数中若抛出异常，不论什么类型，都会被替换为bad_exception类型
bad_typeid	使用typeid操作符，操作一个NULL指针，而该指针是带有虚函数的类，这时抛出bad_typeid异常
bad_cast	使用dynamic_cast转换引用失败的时候
ios_base::failure	io操作过程出现错误
logic_error	逻辑错误，可以在运行前检测的错误
runtime_error	运行时错误，仅在运行时才可以检测的错误

logic_error的子类

异常名称	描述
length_error	试图生成一个超出该类型最大长度的对象时，例如vector的resize操作

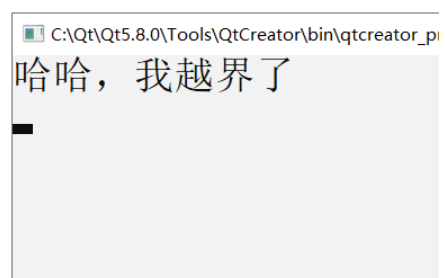
domain_error	参数的值域错误，主要用在数学函数中。例如使用一个负值调用只能操作非负数的函数
outofrange	超出有效范围
invalid_argument	参数不合适。在标准库中，当利用string对象构造bitset时，而string中的字符不是' 0' 或' 1' 的时候，抛出该异常

runtime_error的子类

异常名称	描述
range_error	计算结果超出了有意义的值域范围
overflow_error	算术计算上溢
underflow_error	算术计算下溢
invalid_argument	参数不合适。在标准库中，当利用string对象构造bitset时，而string中的字符不是' 0' 或' 1' 的时候，抛出该异常

```
try
{
    throw out_of_range("哈哈，我越界了");
    //throw bad_alloc();
}
catch(exception &e)
{
    //what()存放的是异常信息（char *方式存在）
    cout<<e.what()<<endl;
}
```

上面的代码 很重要



知识点7【编写自己的异常】（了解）

编写字节的异常：基于标准异常


自己的异常 一定要从exception上继承

```
1 class NewException:public exception
2 {
3 private:
4     string msg;
5 public:
6     NewException(){}
7     NewException(string msg)
8     {
```

```

9   this->msg = msg;
10  }
11  //重写父类的what
12  virtual const char* what()const throw()//防止父类在子类前抛出标准异常
13  {
14      //将string类转换成char *
15      return this->msg.c_str();
16  }
17  ~NewException(){}
18 };
19
20 void test05()
21 {
22     try
23     {
24         throw NewException("哈哈，自己的异常");
25     }
26 }
27 catch(exception &e)
28 {
29     cout<<e.what()<<endl;
30 }
31 }

```

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcra

哈哈，自己的异常