面向对象程序方法与C++程序设计

第7章　异常处理

大连理工大学
主讲人-于红

本章知识点



异常处理

- What 理解异常处理机制
- How 异常处理的实现
- When 异常的类型匹配

异常的类型匹配



- 使用throw语句抛出的异常，可以是基本类型的，也可以用类对象来传递异常信息。
- 专门用来传递异常的类称为异常类（Exception Class）

```
class OutOfBounds{    //数组下标越界异常类
    int index;  // 下标
public:
    OutOfBounds(int index):index(index){}
    void show(){cout<<“数组下标越界异常
    index=”<<index<<endl;}
};
```

每个数组下标越界异常对象中都具有一个数据成员用于描述数组越界访问的具体位置，**catch**块捕捉到这个异常后，可以显示出其包含的越界位置信息。

异常的类型匹配



抛出异常类

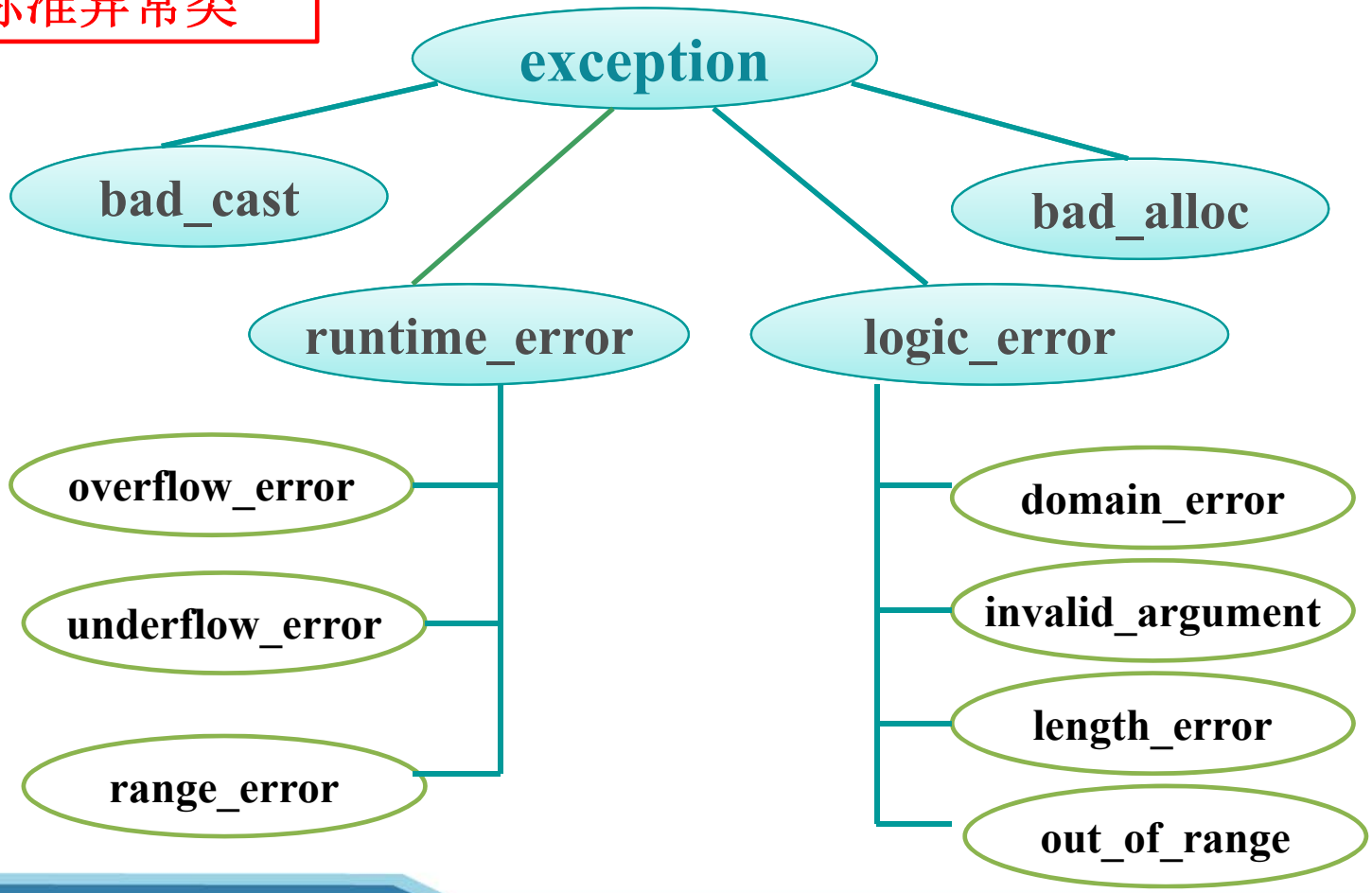
```
class OutOfBounds;
class MyArray{// 数组类
    int *p;
    int sz;
public:
     MyArray(int s);
    ~MyArray( ) ;
     int& operator[ ] (int i){     //重载[]运算符
           if(i<0 || i>=sz) throw OutOfBounds(i);
          return p[i];        }
};
```

使用类对象描述
某种异常的优点？

异常的类型匹配



标准异常类



异常的类型匹配



- 异常类型－基本类型
  - catch参数类型应是异常的类型或其引用；
  - 此时类型必须完全一致，不支持自动类型转换。

最佳匹配

- 异常类型－class类型
  - catch参数类型
    - 异常对象的类型（或其引用）或者其公有基类类型（或其引用）。
    - 该类指针或者其公有基类指针。

最先匹配

- 当catch参数类型为void*时，异常类型可是任何类型指针。
- 当catch参数类型为…时，异常类型可是任何类型。
- catch子句中，较具体的异常捕捉在前，较抽象的异常捕捉在后。

异常的类型匹配



```
#include <iostream>
using namespace std;
int main(){
    for(int i=1;i<=3;i++){
        try{
            if(i==1) throw 0.0;
            if(i==2) throw 0;
            if(i==3) throw 0.0f;
          }
      catch( double ){
          cout<<"Error1: /0.0 "<<endl;}
      catch(int){
          cout<<"Error2: /0 "<<endl;}
       catch(...){
            cout<<"Error3 "<<endl;}
     }
    return 0;
}
```

异常类型必须与**catch**参数类型完全一致，不支持自动类型转换

异常的类型匹配



```
#include <iostream>
using namespace std;
class BaseExcept{ };                    //基类
class DerivExcept:public BaseExcept{ };//派生类
int main(){
    BaseExcept be;                      //基类对象
    DerivExcept de;                     //派生类对象
    BaseExcept *pBase = &be;            //基类指针
    DerivExcept *pDe= &de;              //派生类指针
    cout << "开始" << endl;
    for(int i=1; i<=5; i++){
     try{
        switch(i){
        case 1: cout<<"抛出BaseExcept     ";
                throw be;  break;
        case 2: cout<<"抛出DerivExcept    ";
                throw de; break;
        case 3:cout<<"抛出BaseExcept*     ";
                throw pBase; break;
        case 4: cout<<"抛出DerivExcept*   ";
                throw pDe; break;
         case 5: cout<<"抛出char*    ";
                 throw "excpetion";
        }
    }
    catch(DerivExcept& ){cout<<"匹配DerivExcept"<<endl;}
    catch(BaseExcept& ){cout<<"匹配BaseExcept"<<endl;}
    catch(DerivExcept *){cout<<"匹配DerivExcept*"<<endl;}
    catch(BaseExcept *){cout<<"匹配BaseExcept*"<<endl;}
    catch(void *){cout<<"匹配void*"<<endl;}
    }
    return 0;
    }
```

异常的类型匹配



```
#include <iostream>
using namespace std;
class BaseExcept{ };                    //基类
class DerivExcept:public BaseExcept{ };//派生类
int main(){
    BaseExcept be;                      //基类对象
    DerivExcept de;                     //派生类对象
    BaseExcept *pBase = &be;            //基类指针
    DerivExcept *pDe= &de;              //派生类指针
    cout << "开始" << endl;
    for(int i=1; i<=5; i++){
     try{
        switch(i){
        case 1: cout<<"抛出BaseExcept     ";
                throw be;  break;
        case 2: cout<<"抛出DerivExcept    ";
                throw de; break;
        case 3:cout<<"抛出BaseExcept*     ";
                throw pBase; break;
        case 4: cout<<"抛出DerivExcept*   ";
                throw pDe; break;
        case 5: cout<<"抛出char*    ";
                throw "excpetion";
        }
     }
    catch(BaseExcept& ) {cout<<"匹配BaseExcept"<<endl;}
    catch(DerivExcept& ){cout<<"匹配DerivExcept"<<endl;}
    catch(DerivExcept *){cout<<"匹配DerivExcept*"<<endl;}
    catch(BaseExcept *){cout<<"匹配BaseExcept*"<<endl;}
    catch(void *){cout<<"匹配void*"<<endl;}
    }
    return 0;
}
```

也将捕获派生类异常对象

面向对象程序方法与C++程序设计

第7章　异常处理

大连理工大学
主讲人-于红