通过继承和重载exeception类定义新的异常

exeception类有一个虚函数

virtual const char* what() const; //c++11之后不再用了
virtual const char* what() const noexcept;
	const 后面跟throw() 不是函数，这个东西叫**异常规格说明**，表示 what 函数可以抛出异常的类型，类型说明放到（）里，这里面没有类型，就是声明这个函数不抛出异常，**通常函数不写后面的就表示函数可以抛出任何类型的异常。**

异常规格说明：

1、异常规格说明的目的是为了让函数使用者知道该函数可能抛出的异常有哪些。 可以在函数的声明中列出这个函数可能抛掷的所有异常类型。例如：void fun() throw(A, B, C, D);

2、若无异常接口声明，则此函数可以抛掷任何类型的异常。

3、不抛掷任何类型异常的函数声明如下：void fun() throw();

```
//继承exeception类实现自己的异常
#include<iostream>
#include<exception>
using namespace std;
class MyException:public exception
{
        public:
                const char* what()const throw()
//函数后面必须跟throw(),括号里面不能有任务参数，表示不抛出任务异常
//因为这个已经是一个异常处理信息了，不能再抛异常。
                {
                        return "MyException";
                }
};
int main()
{
        try
        {
                throw MyException();
        }
        catch(MyException & e)    //传引用，防止调用拷贝构造函数
        {
```

```
[meihao@ubuntu ~/learning/06022018]$>./a.out
this is MyException
MyException
```

```
                cout<<"th
is is MyException"<<endl;
                cout<<e.w
hat()<<endl;
        }
}
```

```
//自己实现异常
#include<iostream>
#include<string>
using namespace std;
class exA
{
        public:
                exA(strin
g strA):_strA(strA){ cout
<<"exA()"<<endl; }
                ~exA(){ c
out<<"~exA()"<<endl; }
                virtual c
onst char* what()const th
row()
                {
                        r
eturn _strA.c_str();
                }
        private:
                string _s
trA;
};
class exB : public exA
{
        public:
                exB(strin
g strB):_strB(strB),exA
("exA"){ cout<<"exB()"<<e
ndl; }
                ~exB(){ c
out<<"~exB()"<<endl; }
                const cha
r* what()const throw()
                {
                        r
eturn _strB.c_str();
                }
        private:
                string _s
trB;
};
void fun(int n) throw(in
t,exA,exB)    //只能抛出i
nt,exA,exB类型错误
int main()
{
        try
        {
        //      fun(1);
        //      fun(2);
                fun(3);
    //这里最终调动的是exA
                //因为在
抛出异常对象的时候又建立了
一个exA对象
                //要想执
行异常exB,就要注释掉catch
(exA& b)
                //不注释
掉最后结果就是程序最后面一
个
        //      fun(4);
    //运行出现下列错误
                //termina
te called after throwing
an instance of 'double'
                //Aborted
(core dumped)
        }
        catch(int a)
        {
                cout<<"th
row int"<<endl;
        }
//      catch(exA& b)
//      {
//              cout<<b.w
hat()<<endl;
//              cout<<"th
row exA"<<endl;
//      }
        catch(exB& c)
        {
                cout<<c.w
hat()<<endl;
                cout<<"th
row exB"<<endl;
        }
```

```
{
        if(1==n){throw 1;}
        else if(2==n){throw exA("this is exA");}
        else if(3==n){ throw exB("this is exB");}
        else {throw 2.2;}
}
```

```
        catch(...)
        {
                cout<<"oth exception"<<endl;
        }
        return 0;
}
```

```
//throw int

//exA()
//throw exA
//~exA()

//exA()
//this is exA
//throw exA
//~exA()
```

```
//exA()
//exB()
//this is exB
//throw exB
//~exB()
//~exA()

//exA()
//exB()
//this is exB
//throw exB
//~exB()
//~exA()
```

```
#include<iostream>
#include<exception>
using namespace std;
class myException:public exception
{
        public:
                myException(const char* str):_str(const_cast<char*>(str)) { }
                virtual ~myException()throw() { cout<<"~myException()"<<endl; }
                //析构函数不能抛出异常，不许加异常规格说明throw()
                const char* what()
                {
                        return _str;
                }
        private:
                char* _str;
};
int func(int a ,int b)throw(myException)  //函数可能抛出myException异常
{
        if(0==b)
        {
```

```
                throw *(new myException("除数不能为0"));    //两种都可以
            }
            return a/b;
}
```