以下列举三种对代码中的错误处理方式。

1、对代码中的各个错误进行分别处理。（对不同错误需要进行的相同操作也分开编码）

bool func()

{

代码块;

if(…操作1…)// 操作1执行失败

{

对操作1失败的出错处理;//对操作1失败的特殊处理

公共出错处理; //对所有出错都要进行的处理

return false;

}

if(…操作2…)// 操作2执行失败

{

对操作2失败的出错处理;

公共出错处理;

return false;

}

//所有操作均正确时

当所有操作均正确时执行的代码;

return true;

}

显然这种方式编码会出现冗余，因为对不同错误进行的相同操作是分开编码的（如标红部分）。

2、针对代码中的各个错误的特殊性进行分别处理，对不同错误的共性进行统一处理。

bool func()

{

代码块;

if(…操作1…)//执行失败

{

对操作1失败的出错处理;

goto ERROR\_END;

}

if(…操作2…)//执行失败

{

对操作2失败的出错处理;

goto ERROR\_END;

}

//所有操作均正确时

当所有操作均正确时执行的代码;

return true;

ERROR\_END: //对出错操作的公共处理

对操作1、2失败的公共出错代码(所对某些内存的释放);

return false;

}

显然这种方式编码避免了冗余的错误处理代码，因为对不同错误的共性是统一进行处理的（如标红部分），但是它使用的goto 语句。

3、针对代码中的各个错误的特殊性进行分别处理，对不同错误的共性进行统一处理。它和第二种方法一样，但是它没有使用 goto语句，而是用do-while +break实现。

bool func()

{

代码块;

do{

if(…操作1…)//执行失败

{

对操作1失败的出错处理;

break;

}

if(…操作2…)//执行失败

{

对操作2失败的出错处理;

break;

}

//所有操作均正确时

当所有操作均正确时执行的代码;

return true;

} while (0);

对操作1、2失败的公共出错代码(所对某些内存的释放);

return false;

}