C++语言程序设计

高效编程

宋霜

哈尔滨工业大学（深圳）

机电工程与自动化学院

邮箱: **songshuang@hit.edu.cn**

尽量用const 和inline 而不用#define

- 尽量用编译器而不用预处理。

Effective C++

尽量用const 和inline 而不用#define

- 尽量用编译器而不用预处理。
- #define max(a,b) ((a) > (b) ? (a) : (b))
- int a = 5, b = 0;
- max(++a, b);　　// a 的值增加了?　次
- max(++a, b+10); // a 的值增加了?　次

Effective C++

尽量用const 和inline 而不用#define

- 尽量用编译器而不用预处理。
- #define max(a,b) ((a) > (b) ? (a) : (b))
- int a = 5, b = 0;
- max(++a, b);　　// a 的值增加了?　次
- max(++a, b+10); // a 的值增加了?　次
- template<class T>
- inline const T& max(const T& a, const T& b)
- { return a > b ? a : b; }

Effective C++

尽量用<iostream>而不用<stdio.h>

- scanf 和printf 不是类型安全的，而且没有扩展性。
- 类型安全和扩展性是C++的基石

Effective C++

尽量用<iostream>而不用<stdio.h>

- scanf 和printf 不是类型安全的，而且没有扩展性。
- 类型安全和扩展性是C++的基石

尽量用new 和delete 而不用malloc 和free

- 构造函数和析构函数
- 析构函数里对指针成员调用delete

Effective C++

为需要动态分配内存的类声明一个拷贝构造函数和一个赋值操作符

Effective C++

为需要动态分配内存的类声明一个拷贝构造函数和一个赋值操作符

尽量使用初始化而不要在构造函数里赋值

Effective C++

为需要动态分配内存的类声明一个拷贝构造函数和一个赋值操作符

尽量使用初始化而不要在构造函数里赋值

初始化列表中成员列出的顺序和它们在类中声明的顺序相同

Effective C++

让operator=返回*this 的引用

Effective C++

让operator=返回*this 的引用

在operator=中对所有数据成员赋值

Effective C++

让operator=返回*this 的引用

在operator=中对所有数据成员赋值

在operator=中检查给自己赋值的情况

Effective C++

争取使类的接口完整并且最小

分清成员函数，非成员函数和友元函数

Effective C++

争取使类的接口完整并且最小

分清成员函数，非成员函数和友元函数

- 虚函数必须是成员函数。
- operator>>和operator<<决不能是成员函数
- 只有非成员函数对最左边的参数进行类型转换
- 其它情况下都声明为成员函数

避免public 接口出现数据成员

Effective C++

避免public 接口出现数据成员

尽可能使用const

Effective C++

避免public 接口出现数据成员

尽可能使用const

尽量用“传引用”而不用“传值”

Effective C++

避免public 接口出现数据成员

尽可能使用const

尽量用“传引用”而不用“传值”

必须返回一个对象时不要试图返回一个引用

Effective C++

使公有继承体现 "是一个" 的含义

Effective C++

使公有继承体现 "是一个" 的含义

通过分层来体现 "有一个" 或 "用...来实现"

Effective C++

使公有继承体现 "是一个" 的含义

通过分层来体现 "有一个" 或 "用...来实现"

区分接口继承和实现继承

Effective C++

使公有继承体现 "是一个" 的含义

通过分层来体现 "有一个" 或 "用...来实现"

区分接口继承和实现继承

- 定义纯虚函数的目的在于，使派生类仅仅只是继承函数的接口。
- 声明简单虚函数的目的在于，使派生类继承函数的接口和缺省实现。
- 声明非虚函数的目的在于，使派生类继承函数的接口和强制性实现。

Effective C++

使公有继承体现 "是一个" 的含义

通过分层来体现 "有一个" 或 "用...来实现"

区分接口继承和实现继承

- 定义纯虚函数的目的在于，使派生类仅仅只是继承函数的接口。
- 声明简单虚函数的目的在于，使派生类继承函数的接口和缺省实现。
- 声明非虚函数的目的在于，使派生类继承函数的接口和强制性实现。

决不要重新定义继承而来的非虚函数