第37条：

使用accumulate或者for\_each进行区间统计

例：程序effective\_stl\_test14

例：effective\_stl\_test14

*vector*<float> vf{ 1.2f, 3.5f, 5.7f };

// 容器元素的乘积

// 1.0f为initValue，multiplies<float>()为指定的函数op

// initValue op a1 op a2 op a3...

float product = *accumulate*(vf.*begin*(), vf.*end*(), 1.0f, *multiplies*<float>());

*cout* << "Product : " << product << *endl*; // 23.94

第38条：

设计算法的函数对象时，应该遵循值传递原则。

设计函数对象类时：

1. 函数对象应尽可能的小，减少拷贝的开销。
2. 函数对象类中不应使用虚函数，以避免在出现的“剥离问题（基类类型，传递派生类对象）” ，虚函数无法使用。

如果函数对象类中包含了虚函数，可以将虚函数从函数对象类中分离出来放到一个新类中，然后在函数对象类中包含一个指向新类对象的指针。

// 包含虚函数的函数对象类

template<typename T>

class BPFC : public *unary\_function*<T, void>

{

private:

int x;

public:

virtual void operator() (const T& val0 const)

{}

};

改造后的函数对象类：

template<typename T> class BPFC;

// 分离的新类

template<typename T>

class BPFCImpl : public *unary\_function*<T, void>

{

private:

int x;

virtual ~BPFCImpl()

{}

virtual void operator()(const T& val) const

{

}

friend class BPFC<T>;

};

template<typename T>

class BPFC : public *unary\_function*<T, void>

{

private:

BPFCImpl<T>\* impl\_ptr\_;

public:

void operator() (const T& val) const

{

impl\_ptr\_->operator()(val);

}

};