第7条：

STL容器中包含了通过new创建的指针，一定要在容器对象析构前将指针delete掉。

例：程序effective\_stl\_test3

class Test

{

public:

~Test()

{

*cout* << "Destruct Test" << *endl*;

}

};

// 函数执行完后，不会调用Test的析构函数

// 会造成内存泄漏

void DoSomething()

{

Test\* ptr = new Test();

*vector*<Test\*> vec1;

vec1.*push\_back*(ptr);

}

补救措施：

// 函数对象

struct DeleteObject

{

template<typename T>

void operator() (const T\* ptr) const

{

delete ptr;

}

};

void DoSomething1()

{

Test\* ptr = new Test();

*vector*<Test\*> vec1;

vec1.*push\_back*(ptr);

*for\_each*(vec1.*begin*(), vec1.*end*(), DeleteObject());

}

调用DoSomething1()，会调用Test的析构函数

Destruct Test

这种情况，在调用for\_each之前发生异常，则还是会造成内存泄漏。

最安全的办法是使用智能指针：

例：程序effective\_stl\_test3

void DoSomething2()

{

typedef *shared\_ptr*<Test> SPTest;

Test\* ptr = new Test();

SPTest smart(ptr);

*vector*<SPTest> vec1;

vec1.*push\_back*(smart);

}

调用DoSomething2()结束后，会自动调用Test的析构函数

第8条：

不要创建包含auto\_ptr的容器对象。

第9条：

容器元素的删除：

1. 删除容器中有特定值的所有对象：

如果容器是vector、string或deque，使用erase-remove习惯用法

如果容器是list，使用list::remove。

如果容器时关联容器，使用它的erase成员函数。

1. 删除容器中满足特定判别式的所有对象

如果容器是vector、string或deque，使用erase-remove\_if习惯用法

如果容器是list，使用list::remove\_if。

如果容器时关联容器，使用下面的方法：

AssocContainer<int> c;

for (AssocContainer<int>::iterator i = c.begin(); i != c.end();)

{

if (BadValue(\*i))

c.erase(i++);

else

++i;

}

第10条：

第11条：

分配器allocator

第12条：

C++ STL容器不是线程安全的。