1、**RAII**(Resource Acquisition Is Initialization)，**RAII**的基本含义就是：C++中的资源（例如内存，文件句柄等等）应该由对象来管理，资源在对象的构造函数中初始化，并在对象的析构函数中被释放。其中一个经典的应用就是智能指针。

应用实例链接：

1、多线程同步之区域锁：http://blog.csdn.net/zouxinfox/article/details/5848519

区域锁就是把锁封装到一个对象里面。锁的初始化放到构造函数，锁的释放放到析构函数。这样当锁离开作用域时，析构函数会自动释放锁。即使运行时抛出异常，由于析构函数仍然会自动运行，所以锁仍然能自动释放。一个典型的区域锁：

class Thread\_Mutex\_Guard

{

public:

Thread\_Mutex\_Guard (Thread\_Mutex &lock)

: lock\_ (&lock)

{

// 如果加锁失败，则返回-1

owner\_= lock\_->lock();

}

~Thread\_Mutex\_Guard (void)

{

// 如果锁获取失败，就不释放

if (owner\_ != -1)

lock\_->unlock ();

}

private:

Thread\_Mutex \*lock\_;

int owner\_;

};

改进版本：在不需要使用的时候，可以手动释放锁

class Thread\_Mutex\_Guard

{

public:

Thread\_Mutex\_Guard (Thread\_Mutex &lock)

: lock\_ (&lock)

{

acquire();

}

int acquire()

{

// 加锁失败，返回-1

owner\_= lock\_->lock();

return owner;

}

~Thread\_Mutex\_Guard (void)

{

release();

}

int release()

{

// 第一次释放

if (owner\_ != -1)

{

owner = -1;

return lock\_->unlock ();

}

// 第二次释放

return 0;

}

private:

Thread\_Mutex \*lock\_;

int owner\_;

};

### 缺点：

区域锁固然好使，但也有不可避免的一些缺点

(1) 对于[非递归锁](http://blog.csdn.net/zouxinfox/archive/2010/08/25/5838861.aspx" \t "_blank)，有可能因为重复加锁而造成死锁。

(2) 线程的强制终止或者退出，会造成区域锁不会自动释放。应该尽量避免这种情形，或者使用一些特殊的错误处理设计来确保锁会释放。

(3) 编译器会产生警告说有变量只定义但没有使用。有些编译器选项甚至会让有警告的程序无法编译通过。在ACE中，为了避免这种情况，作者定义了一个宏如下

#define UNUSED\_ARG(arg) { if (&arg) /\* null \*/; }

Thread\_Mutex\_Guard guard (lock\_);

UNUSED\_ARG (guard);