c++经验之谈一：RAII原理介绍

allen

1.什么是RAII

RAII（**R**esource **A**cquisition **I**s **I**nitialization）是由c++之父Bjarne Stroustrup提出的，中文翻译为资源获取即初始化，他说：使用局部对象来管理资源的技术称为资源获取即初始化；这里的资源主要是指操作系统中有限的东西如内存、网络套接字等等，局部对象是指存储在栈的对象，它的生命周期是由操作系统来管理的，无需人工介入；

2.RAII的原理

资源的使用一般经历三个步骤a.获取资源 b.使用资源 c.销毁资源，但是资源的销毁往往是程序员经常忘记的一个环节，所以程序界就想如何在程序员中让资源自动销毁呢？c++之父给出了解决问题的方案：RAII，它充分的利用了C++语言局部对象自动销毁的特性来控制资源的生命周期。给一个简单的例子来看下局部对象的自动销毁的特性：

```
#include <iostream>
using namespace std;
class person {
  public:
      person(const std::string name = "", int age = 0) :
      name_(name), age_(age) {
            std::cout << "Init a person!" << std::endl;
      }
      ~person() {
            std::cout << "Destory a person!" << std::endl;
      }
      const std::string& getname() const {
            return this->name_;
      }
      int getage() const {
            return this->age_;
      }
  private:
      const std::string name_;
      int age_;
};
int main() {
    person p;
    return 0;
}
编译并运行:
g++ person.cpp -o person
./person
运行结果:
Init a person!
Destory a person!
```

从person class可以看出，当我们在main函数中声明一个局部对象的时候，会自动调用构造函数进行对象的初始化，当整个main函数执行完成后，自动调用析构函数来销毁对象，整个过程无需人工介入，由操作系统自动完成；于是，很自然联想到，当我们在使用资源的时候，在构造函数中进行初始化，在析构函数中进行销毁。整个RAII过程我总结四个步骤：

a.设计一个类封装资源

b.在构造函数中初始化

c.在析构函数中执行销毁操作

d.使用时声明一个该对象的类

3.RAII的应用

本节主要通过一个简单的例子来说明如何将RAII应用到我们的代码中。linux下经常会使用多线程技

资源一次只被一个线程访问，按照我们前面的分析，我们封装一下POSIX标准的互斥锁：

```
#include <pthread.h>
#include <cstdlib>
#include <stdio.h>

class Mutex {
 public:
  Mutex();
  ~Mutex();

  void Lock();
  void Unlock();

 private:
  pthread_mutex_t mu_;

  // No copying
  Mutex(const Mutex&);
  void operator=(const Mutex&);
};


#include "mutex.h"

static void PthreadCall(const char* label, int result) {
  if (result != 0) {
    fprintf(stderr, "pthread %s: %s\n", label, strerror(result));
  }
}

Mutex::Mutex() { PthreadCall("init mutex", pthread_mutex_init(&mu_, NULL)); }

Mutex::~Mutex() { PthreadCall("destroy mutex", pthread_mutex_destroy(&mu_)); }

void Mutex::Lock() { PthreadCall("lock", pthread_mutex_lock(&mu_)); }

void Mutex::Unlock() { PthreadCall("unlock", pthread_mutex_unlock(&mu_)); }
```

写到这里其实就可以使用Mutex来锁定临界区，但我们发现Mutex只是用来对锁的初始化和销毁，我们还得在代码中调用Lock和Unlock函数，这又是一个对立操作，所以我们可以继续使用RAII进行封装，代码如下：

```
#include "mutex.h"

class  MutexLock {
 public:
  explicit MutexLock(Mutex *mu)
      : mu_(mu)  {
    this->mu_->Lock();
  }
  ~MutexLock() { this->mu_->Unlock(); }

 private:
  Mutex *const mu_;
  // No copying allowed
  MutexLock(const MutexLock&);
  void operator=(const MutexLock&);
};
```

到这里我们就真正封装了互斥锁，下面我们来通过一个简单的例子来使用它，代码如下：

```
#include "mutexlock.hpp"
#include <unistd.h>
#include <iostream>

#define    NUM_THREADS     10000

int num=0;
Mutex mutex;

void *count(void *args) {
    MutexLock lock(&mutex);
    num++;
}


int main() {
    int t;
    pthread_t thread[NUM_THREADS];

    for( t = 0; t < NUM_THREADS; t++) {
        int ret = pthread_create(&thread[t], NULL, count, NULL);
        if(ret) {
            return -1;
        }
    }

    for( t = 0; t < NUM_THREADS; t++)
        pthread_join(thread[t], NULL);
    std::cout << num << std::endl;
```

```
编译并运行：g++ test_mutexlock.cpp mutexlock.hpp mutex.cpp mutex.h -o test_mutexl
./test_mutexlock
运行结果：10000  符合预期（可以去掉MutexLock lock(&mutex);试试看看结果  何？）
```

jiayunwei/mutexLock
github.com/jiayunwei/mutexLock

编辑于 2018-06-11 22:03

内容所属专栏

c++：从入门到放弃 订阅专栏

C++ 面向对象编程

写下你的评论...

60 条评论 默认 最新

不知道该叫什么
所以说RAII机制是一种对资源申请、释放这种成对的操作的封装是么 通过这种方式实现在局部作用域内申请资源然后销毁资源呗？

2021-04-21 回复 52

allen 作者
是的，总结的不错
2021-07-12 回复 5

barcelona
C++11里面，防止拷贝应该把拷贝构造函数和拷贝赋值运算符设为删除的
2021-01-05 回复 27

BACKXMQ4A ‣ **NULL**
rust迷整天就知道提，rust说的好像这些特性只有rust有似的
2023-05-26 回复 8

allen 作者
是的，说的很对
2021-07-12 回复 7

展开其他 2 条回复 》

毛豆花生
不是操作系统帮你析构，是编译器
2021-07-02 回复 20

Glider ‣ **白马非兔**
可执行文件里已经加上了返回时要析构对象的代码了，所以是编译帮忙做的
2023-01-26 回复 6

VizXu
是语言特性，如果C语言加上一个特性使用一个struct时先执行一个回调函数，struct离开作用域时又执行一个回调函数，那么C语言也可以创造出RAII技术
2022-01-14 回复 4

查看全部 14 条回复 》

Micro
没想到日常创建对象还有这么高深的设计思想？？？🙏
2021-04-22 回复 5

我啥都不知道
Person p; 这种形式在C++中是在栈空间进行内存分配吧，会自动进行回收，所以类似于智能指针这种的实现应该就是利用在栈空间实例化的sharedptr 类来管理指针的释放吗
2021-08-23 回复 6

allen 作者
是的，应该说智能指针就是用这种技术，当然sharedptr还有引用计数技术
2021-08-28 回复 4

George
智能指针是其中一种吧
2023-02-17 回复 5

allen 作者
是的👍
2023-02-26 回复 1

柒贰肆
这个自己封装的和STL里lock guard是一样的嘛
2023-01-20 回复 5

载土而归
是的
2023-08-24 回复 喜欢

▲ 赞同 446 ▼ 60 条评论 分享 喜欢 收藏 申请转载 ···

为什么要用指针，不用引用？
2019-07-24 回复 2

allen 作者
你说的是哪里？
2019-07-24 回复 2

楚沫 ‣ **allen**
应该是说的是那个互斥锁类里面写Mutex *const的是指针常量，为什么不用引用？
2024-05-19 回复 喜欢

时年
可以理解成我们想控制一个对象的生成与回收只需要把这个对象封装在另一个类的构造和析构函数中就可以实现局部的自动内存申请与回收了吗（）

2023-08-05 回复 1

时年 ‣ **allen**
❤️感谢解答，爱您。

2023-08-06 回复 喜欢

allen 作者
是的👍可以这样理解
2023-08-05 回复 喜欢

风清扬
为什么两个测试文件都不能通过编译啊😳😳
2023-02-23 回复 1

点击查看全部评论 ›

写下你的评论...

C++: RAII原理、应用与实践——应该使用对象来管理资源
七昂的技术之旅

C++经验之谈二：RAII的运用
allen

C++中的RAII技术及典型应用
可吉拉多 发表于面向加薪编...

TeX 宏编程技巧之定界的宏参数
众所周知，在 TeX 中定义宏时，宏参数可以使用 tokens 来定界，从而减少宏层面的判断（如 \if 、\ifx 、\strcmp ），以此加快编译时间，LaTeX3 源码中就包含大量的此类技巧，诸如 map 和 br…
知乎用户WY8444

▲ 赞同 446 ▼ 60 条评论 分享 喜欢 收藏 申请转载 …