

```
RAII() : data(nullptr) {}
explicit RAII(T* rhs) : data{ rhs } {}
~RAII() {
    if(data) delete data;
}

T* operator ->() const {
    return data;
}
```

现代C++学习—— 什么是RAII

 严格鸽 
柚子厨/萝莉控/acm银

115 人赞同了该文章

我们将内存简单的分为栈内存和堆内存。

栈上的内存不需要考虑释放问题，而堆内存需要手动释放。一般来讲是以下的流程。

```
A* p = new A();

// do something

delete p;
```

- 获取资源
- 使用资源
- 释放资源

通常我们会记得前两步，而忘记第三步，释放资源。

当然可以说，new 一定要和delete配对。

但是在某些情况下，我们会提前跳出，然后忘记匹配一个delete

```
void foo(int n) {
    A* p = new A();

    if (n > 1) {
        /* ... */
        return;
    }

    delete p;
}
```

当你处理错误，或者一些其它的事情，需要提前结束函数/作用域，这个时候，为了保证资源的释放，你需要每一个分支都加个delete。

这样是比较麻烦的，所以这里引入RAII。

RAII，全称资源获取即初始化

RAII要求，资源的有效期与持有资源的对象的生命期严格绑定，即由对象的构造函数完成资源的分配（获取），同时由析构函数完成资源的释放。在这种要求下，只要对象能正确地析构，就不会出现资源泄漏问题。

emmmm，简单来说就是，无论怎么样，结束函数都会调用析构函数，所以我们在析构函数里面写delete。

也就是让编译器在结束函数的每一个分支上，加上析构函数，而我们在析构函数里面delete。

简单的写一下就是这个样子

```
struct Raii{
    A* p;
    Raii(A* _p) : p{ _p } {};
    ~Raii() {
        delete p;
    }
};

void foo() {
    Raii ptr(new A());
}
```


这样析构函数会帮我们自动释放的。

ok，但是我们必然不会为了每一个类都去写一个RAII的管理类，所以用模板来生成吧。

登录即可查看 超5亿 专业优质内容

超 5 千万创作者的优质提问、专业回答、深度文章和精彩视频尽在知乎。

立即登录/注册



```
class RAII {
private:
    T* data;
public:
    RAII() : data(nullptr) {}
    explicit RAII(T* rhs) : data{ rhs } {};
    ~RAII() {
        delete data;
    }
    T* operator ->()const {
        return data;
    }
};
```

这里重载了 -> 来方便我们调用

```
struct A {
    int x, y, z;
    ~A() {
        cout << "~A()" << endl;
    }
};
```

这样使用

```
RAII<A> ptr(new A{ 114,514,1919810 });
cout << ptr->x << endl;
```

果只是为了不要忘记释放资源，上面就够了，但是这里我们需要引入**所有权**的概念

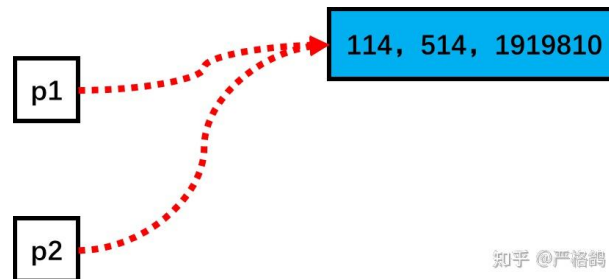
果你对所有权没有概念，推荐阅读 [严格鸽：现代C++学习 —— 为什么需要std::move](#)

其实下面的内容就和**unique_ptr**一样了。

首先，我们这种写法，是独占所有权的。

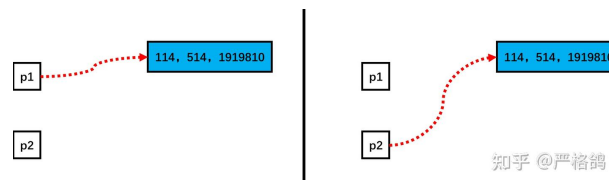
```
RAII<A> p1(new A{ 114,514,1919810 });
RAII<A> p2 = p1;
```

上面这种情况，可以理解为，p1,p2同时有了 A 的所有权。



当我们析构函数释放的时候，显然这个内存，会被我们释放两次。

所以，我们需要用**move**来移交所有权。



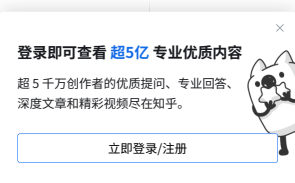
在加上一些东西吧

```
template<typename T>
class RAII {
private:
    T* data;
public:
    RAII() : data(nullptr) {}
    explicit RAII(T* rhs) : data{ rhs } {};

    ~RAII() {
        if(data)delete data;
    }
    T* operator ->()const {
        return data;
    }

    RAII(const RAII<T>&) = delete;
    RAII(RAII<T>&& rhs) {
```

▲ 赞同 115 ▼ ● 5 条评论 🔊 分享 ❤️ 喜欢 ★ 收藏 📄 申请转载 ...



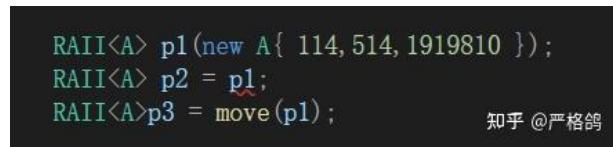
```
        rhs.data = nullptr;
    }
    RAII& operator = (const RAII&) = delete;

    void operator = (RAII<T>&& rhs) {
        data = rhs.data;
        rhs.data = nullptr;
    }
};
```

这里，我们把有关拷贝的函数都删除了。



但是我们可以通过move来移交所有权。



这样的代码

```
RAII<A> p1(new A{ 114, 514, 1919810 });
cout << "here" << endl;
```

输出

```
here
~A()
```

在析构函数在作用域的最后

而这样的代码

```
RAII<A> p1(new A{ 114, 514, 1919810 });
{
    RAII<A> p3 = move(p1);
}
cout << "here" << endl;
```

输出

```
~A()
here
```

我们利用move，把所有权交给了{}里面的一个RAII类，缩窄了其生命周期。

好，那么问题又来了， 何把RAII类作为函数的参数？

这里有个回答就很好

[zhihu.com/question/5343...](https://www.zhihu.com/question/5343...)



郭宽
数据存储

14 人赞同了该回答

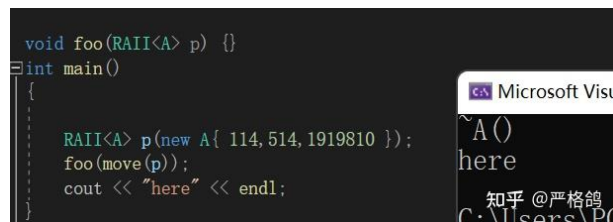
值传递：各位caller，我不要ownership了，请拿走

非const 引用传递：你拿不拿走都行，提前跟我商量好（不推荐）

const 引用传递：可以拿走用一下，ownership还是我的

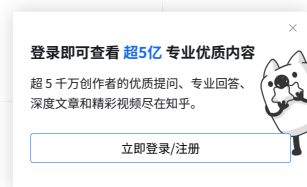
右值引用：同第二条，无法确定caller是否拿走了ownership 知乎 @严格鸽

看情况使用第一条和第三条



```
void foo(const RAII<A> &n) {}
```

赞同 115 5 条评论 分享 喜欢 收藏 申请转载 ...



```
RAII{A} p(new A{ 114, 514, 1919810 });
foo(p);
cout << "here" << endl;
```

发布于 2023-01-21 09:17 · IP 属地山东

内容所属专栏

C++模板学习实践

订阅专栏

C++ C/C++ Modern C++

写下你的评论...

- 5 条评论
- 默认排序 最新
- 滕滕滕

言简意赅

2024-04-06 · 江苏

回复 喜欢
- 实名用户1

有点味道的教程

2023-12-10 · 广东

回复 喜欢
- 风zx

写的很好

2023-09-27 · 北京

回复 喜欢
- l-Taozi

2023-03-16 · 上海

回复 喜欢
- 腾腾

好好好好好

2023-02-02 · 广西

回复 喜欢

推荐阅读

MANCHESTER 1824

the University of Manchester

C/C++ segment fault (core dumped)

张树刚

C++

我最常用的C++编程技法

cpp编程

C++23特性总结 - 下

Mick235711

基础篇：struct, class和 union

目录：C结构体、C++结构体和 C++类的区别回顾：从C的POD到C++的类C++结构体和类总结C++类所占内存大小C++中的union想要回顾一下基本概念，请参考：类和结构(C++)C结构体、C++结构体和...

咖啡 发表于Moder...

登录即可查看 超5亿 专业优质内容

超 5 千万创作者的优质提问、专业回答、深度文章和精彩视频尽在知乎。

立即登录/注册

赞同 115 5 条评论 分享 喜欢 收藏 申请转载