



首页 / 正文

转发

微博

Qzone

微信

现代 C++ 一文读懂智能指针

闪念基因 2020-06-19 09:50:33

智能指针

C++11 引入了 3 个智能指针类型：

1. `std::unique_ptr<T>`：独占资源所有权的指针。
2. `std::shared_ptr<T>`：共享资源所有权的指针。
3. `std::weak_ptr<T>`：共享资源的观察者，需要和 `std::shared_ptr` 一起使用，不影响资源的生命周期。

`std::auto_ptr` 已被废弃。

`std::unique_ptr`

简单说，当我们独占资源的所有权的时候，可以使用 `std::unique_ptr` 对资源进行管理——离开 `unique_ptr` 对象的作用域时，会自动释放资源。这是很基本的 RAII 思想。

`std::unique_ptr` 的使用比较简单，也是用得比较多的智能指针。这里直接看例子。

1. 使用裸指针时，要记得释放内存。

```
{
    int* p = new int(100);
    // ...
    delete p; // 要记得释放内存
}
```

2. 使用 `std::unique_ptr` 自动管理内存。

```
{
    std::unique_ptr<int> uptr = std::make_unique<int>(200);
    // ...
    // 离开 uptr 的作用域的时候自动释放内存
}
```

3. `std::unique_ptr` 是 move-only 的。

```
{
    std::unique_ptr<int> uptr = std::make_unique<int>(200);
    std::unique_ptr<int> uptr1 = uptr; // 编译错误，std::unique_ptr<T> 是 move-only

    std::unique_ptr<int> uptr2 = std::move(uptr);
    assert(uptr == nullptr);
}
```

4. `std::unique_ptr` 可以指向一个数组。

```
{
    std::unique_ptr<int[]> uptr = std::make_unique<int[]>(10);
}
```



闪念基因

+关注

连接跟踪：原理及Linux 内核

Service Mesh中的iptables

Go Hash

科普一下跨端

电信运营商用户隐私保护探索



```
for (int i = 0; i < 10; i++) {
    uptr[i] = i * i;
}
for (int i = 0; i < 10; i++) {
    std::cout << uptr[i] << std::endl;
}
}
```

5. 自定义 deleter。

```
{
    struct FileCloser {
        void operator()(FILE* fp) const {
            if (fp != nullptr) {
                fclose(fp);
            }
        }
    };
    std::unique_ptr<FILE, FileCloser> uptr(fopen("test_file.txt", "w"));
}
```

6. 使用 Lambda 的 deleter。

```
{
    std::unique_ptr<FILE, std::function<void(FILE*)>> uptr(
        fopen("test_file.txt", "w"), [](FILE* fp) {
            fclose(fp);
        });
}
```

std::shared_ptr

std::shared_ptr 其实就是对资源做引用计数——当引用计数为 0 的时候，自动释放资源。

```
{
    std::shared_ptr<int> sptr = std::make_shared<int>(200);
    assert(sptr.use_count() == 1); // 此时引用计数为 1
    {
        std::shared_ptr<int> sptr1 = sptr;
        assert(sptr.get() == sptr1.get());
        assert(sptr.use_count() == 2); // sptr 和 sptr1 共享资源，引用计数为 2
    }
    assert(sptr.use_count() == 1); // sptr1 已经释放
}
// use_count 为 0 时自动释放内存
```

和 unique_ptr 一样，shared_ptr 也可以指向数组和自定义 deleter。

```
{
    // C++20 才支持 std::make_shared<int[]>
    // std::shared_ptr<int[]> sptr = std::make_shared<int[]>(100);
    std::shared_ptr<int[]> sptr(new int[10]);
    for (int i = 0; i < 10; i++) {
```



```

        sptr[i] = i * i;
    }
    for (int i = 0; i < 10; i++) {
        std::cout << sptr[i] << std::endl;
    }
}

{
    std::shared_ptr<FILE> sptr(
        fopen("test_file.txt", "w"), [](FILE* fp) {
            std::cout << "close " << fp << std::endl;
            fclose(fp);
        });
}

```

std::shared_ptr 的实现原理

一个 shared_ptr 对象的内存开销要比裸指针和无自定义 deleter 的 unique_ptr 对象略大。

```

std::cout << sizeof(int*) << std::endl; // 输出 8
std::cout << sizeof(std::unique_ptr<int>) << std::endl; // 输出 8
std::cout << sizeof(std::unique_ptr<FILE, std::function<void(FILE*)>>)
    << std::endl; // 输出 40

std::cout << sizeof(std::shared_ptr<int>) << std::endl; // 输出 16
std::shared_ptr<FILE> sptr(fopen("test_file.txt", "w"), [](FILE* fp) {
    std::cout << "close " << fp << std::endl;
    fclose(fp);
});
std::cout << sizeof(sptr) << std::endl; // 输出 16

```

无自定义 deleter 的 unique_ptr 只需要将裸指针用 RAII 的手法封装好就行，无需保存其它信息，所以它的开销和裸指针是一样的。如果有自定义 deleter，还需要保存 deleter 的信息。

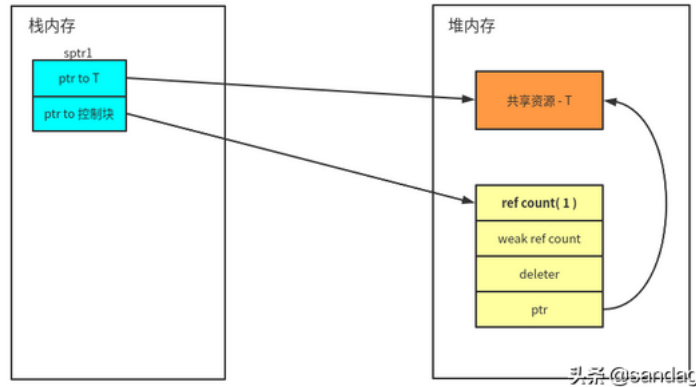
shared_ptr 需要维护的信息有两部分：

1. 指向共享资源的指针。
2. 引用计数等共享资源的控制信息——实现上是维护一个指向控制信息的指针。

所以，shared_ptr 对象需要保存两个指针。shared_ptr 的 deleter 是保存在控制信息中，所以，是否有自定义 deleter 不影响 shared_ptr 对象的大小。

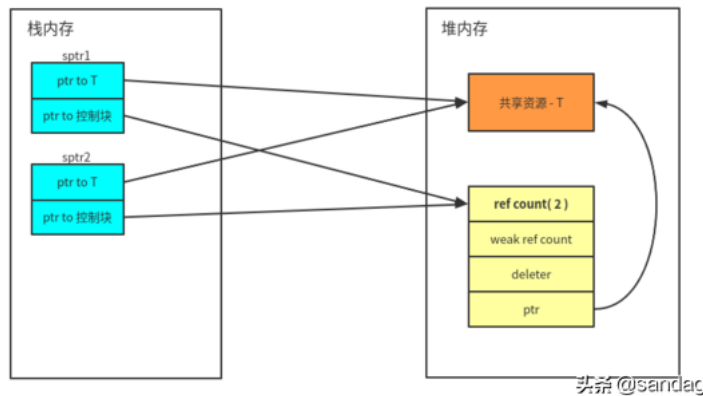
当我们创建一个 shared_ptr 时，其实现一般如下：

```
std::shared_ptr<T> sptr1(new T);
```



复制一个 shared_ptr :

```
std::shared_ptr<T> sptr2 = sptr1;
```



为什么控制信息和每个 shared_ptr 对象都需要保存指向共享资源的指针？可不可以去掉 shared_ptr 对象中指向共享资源的指针，以节省内存开销？

答案是：不能。因为 shared_ptr 对象中的指针指向的对象不一定和控制块中的指针指向的对象一样。

来看一个例子。

```
struct Fruit {
    int juice;
};

struct Vegetable {
    int fiber;
};

struct Tomato : public Fruit, Vegetable {
    int sauce;
};

// 由于继承的存在, shared_ptr 可能指向基类对象
std::shared_ptr<Tomato> tomato = std::make_shared<Tomato>();
std::shared_ptr<Fruit> fruit = tomato;
std::shared_ptr<Vegetable> vegetable = tomato;
```

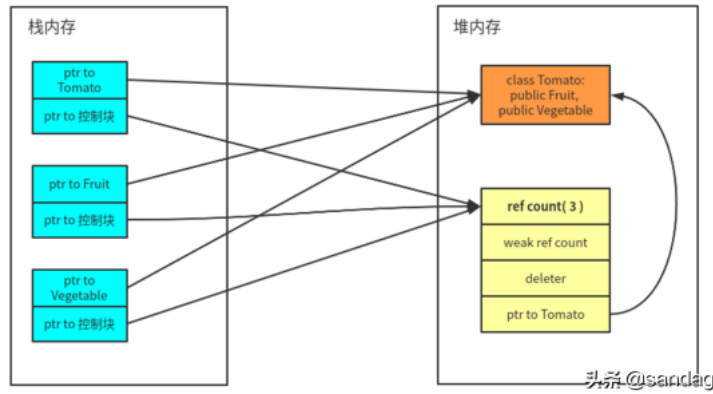
50

转发

微博

Qzone

微信



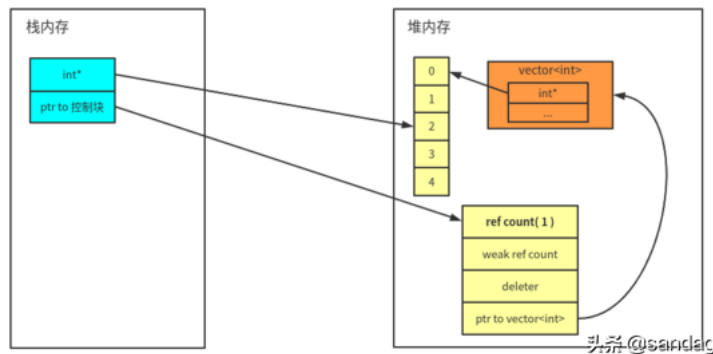
另外，std::shared_ptr 支持 aliasing constructor。

```
template< class Y >
shared_ptr( const shared_ptr<Y>& r, element_type* ptr ) noexcept;
```

Aliasing constructor, 简单说就是构造出来的 shared_ptr 对象和参数 r 指向同一个控制块 (会影响 r 指向的资源的生命周期), 但是指向共享资源的指针是参数 ptr。看下面这个例子。

```
using Vec = std::vector<int>;
std::shared_ptr<int> GetSPtr() {
    auto elts = {0, 1, 2, 3, 4};
    std::shared_ptr<Vec> pvec = std::make_shared<Vec>(elts);
    return std::shared_ptr<int>(pvec, &(*pvec)[2]);
}

std::shared_ptr<int> sptr = GetSPtr();
for (int i = -2; i < 3; ++i) {
    printf("%d\n", sptr.get()[i]);
}
```



看上面的例子, 使用 std::shared_ptr 时, 会涉及两次内存分配: 一次分配共享资源对象; 一次分配控制块。C++ 标准库提供了 std::make_shared 函数来创建一个 shared_ptr 对象, 只需要一次内存分配。



转发



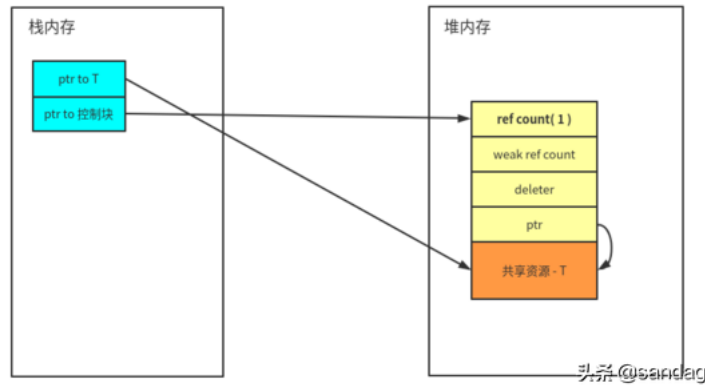
微博



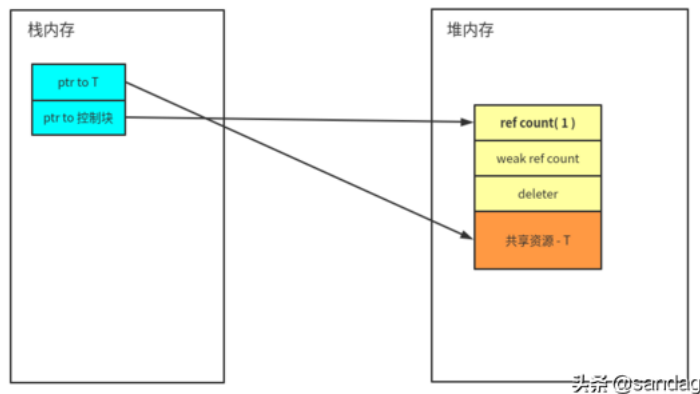
Qzone



微信



这种情况下，不用通过控制块中的指针，我们也能知道共享资源的位置——这个指针也可以省略掉。



std::weak_ptr

std::weak_ptr 要与 std::shared_ptr 一起使用。一个 std::weak_ptr 对象看做是 std::shared_ptr 对象管理的资源的观察者，它不影响共享资源的生命周期：

1. 如果需要使用 weak_ptr 正在观察的资源，可以将 weak_ptr 提升为 shared_ptr。
2. 当 shared_ptr 管理的资源被释放时，weak_ptr 会自动变成 nullptr。

```
void Observe(std::weak_ptr<int> wptr) {
    if (auto sptr = wptr.lock()) {
        std::cout << "value: " << *sptr << std::endl;
    } else {
        std::cout << "wptr lock fail" << std::endl;
    }
}

std::weak_ptr<int> wptr;
{
    auto sptr = std::make_shared<int>(111);
    wptr = sptr;
    Observe(wptr); // sptr 指向的资源没被释放, wptr 可以成功提升为 shared_ptr
}
Observe(wptr); // sptr 指向的资源已被释放, wptr 无法提升为 shared_ptr
```

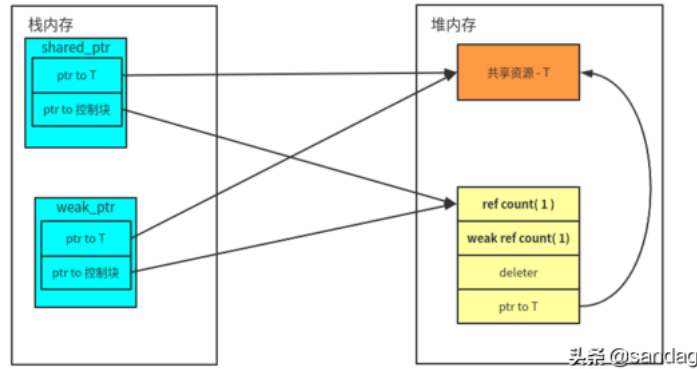
50

转发

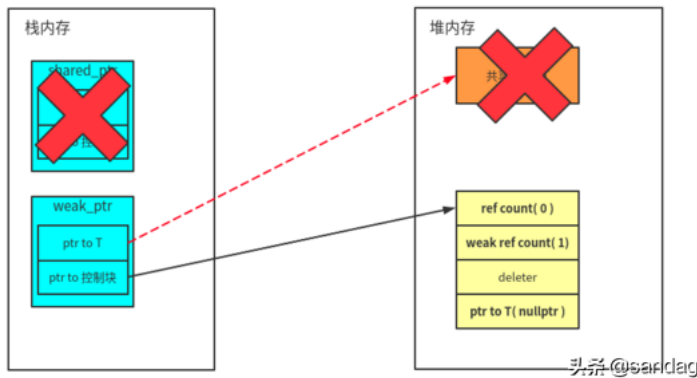
微博

Qzone

微信



当 shared_ptr 析构并释放共享资源的时候，只要 weak_ptr 对象还存在，控制块就会保留，weak_ptr 可以通过控制块观察到对象是否存活。



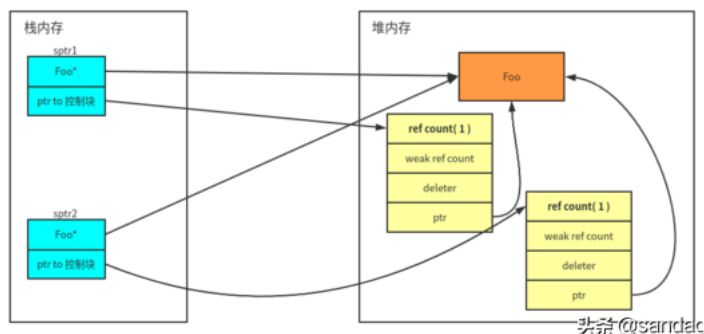
enable_shared_from_this

一个类的成员函数如何获得指向自身 (this) 的 shared_ptr? 看看下面这个例子有没有问题?

```
class Foo {
public:
    std::shared_ptr<Foo> GetSPtr() {
        return std::shared_ptr<Foo>(this);
    }
};

auto sptr1 = std::make_shared<Foo>();
assert(sptr1.use_count() == 1);
auto sptr2 = sptr1->GetSPtr();
assert(sptr1.use_count() == 1);
assert(sptr2.use_count() == 1);
```

上面的代码其实会生成两个独立的 shared_ptr，他们的控制块是独立的，最终导致一个 Foo 对象会被 delete 两次。





转发



微博



Qzone



微信

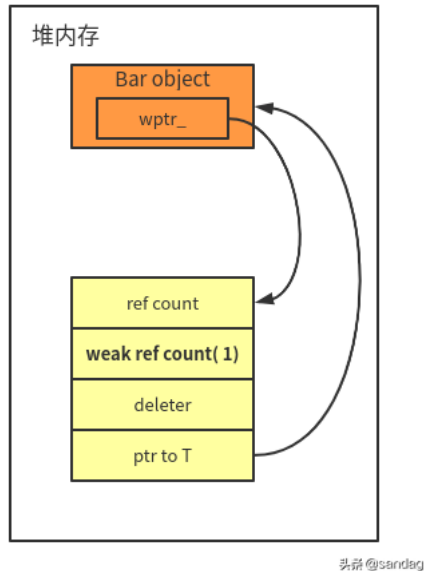
成员函数获取 this 的 shared_ptr 的正确的做法是继承 std::enable_shared_from_this。

```
class Bar : public std::enable_shared_from_this<Bar> {
public:
    std::shared_ptr<Bar> GetSPtr() {
        return shared_from_this();
    }
};

auto sptr1 = std::make_shared<Bar>();
assert(sptr1.use_count() == 1);
auto sptr2 = sptr1->GetSPtr();
assert(sptr1.use_count() == 2);
assert(sptr2.use_count() == 2);
```

一般情况下，继承了

std::enable_shared_from_this 的子类，成员变量中增加了一个指向 this 的 weak_ptr。这个 weak_ptr 在第一次创建 shared_ptr 的时候会被初始化，指向 this。



似乎继承了

std::enable_shared_from_this 的类都被强制必须通过 shared_ptr 进行管理。

```
auto b = new Bar;
auto sptr = b->shared_from_this();
```

在我的环境下 (gcc 7.5.0) 上面的代码执行的时候会直接 coredump，而不是返回指向 nullptr 的 shared_ptr：

```
terminate called after throwing an instance of 'std::bad_weak_ptr'
what(): bad_weak_ptr
```

小结

智能指针，本质上是对资源所有权和生命周期管理的抽象：

1. 当资源是被独占时，使用 std::unique_ptr 对资源进行管理。
2. 当资源会被共享时，使用 std::shared_ptr 对资源进行管理。



- 3. 使用 `std::weak_ptr` 作为 `std::shared_ptr` 管理对象的观察者。
- 4. 通过继承 `std::enable_shared_from_this` 来获取 `this` 的 `std::shared_ptr` 对象。

参考资料

- 1. Back to Basics: Smart Pointers
- 2. cppreference: `unique_ptr`
- 3. cppreference: `shared_ptr`
- 4. cppreference: `weak_ptr`
- 5. cppreference: `enable_shared_from_this`

☆ 收藏 ⓘ 举报

50 条评论

写下您的评论...

评论

Papst 5月前
五六年没看新的c++内容，现在看你这篇文章都不认识这门语言了
[回复 · 2条回复](#)

0 ⓘ

真放过羊的娃 9月前
智能指针 出问题特别难查，不如自己用一般的指针
[回复 · 3条回复](#)

6 ⓘ

只想简单28 9天前
智能指针虽说优点不用自己管理内存了，但使用起来很费劲，有时不好管理，出错了，不方便查错，c++本来就是讲效率的，智能指针增加内存开销，那还有什么好处，那还不如学习其他语言了
[回复](#)

2 ⓘ

meYoda 9月前
写得很好加油
[回复](#)

0 ⓘ

liangX 9月前
干练而无多余废话
[回复](#)

0 ⓘ






[查看更多评论](#)


相关推荐

美卫生官员：美国疫情发展依然令人担忧
视频 ⓘ 央视新闻 · 309评论 · 15分钟前

台湾制造远洋船下水才10秒就侧翻 欢呼声瞬间变惊呼声
视频 ⓘ 看看新闻 · 1万评论 · 30分钟前

学习直播教学工具，直播免费，收益0分成，老师免费入驻中!
[查看详情 >](#)


-  50
-  转发
-  微博
-  Qzone
-  微信

 工人日报

未关注 · 工人日报官方账号

【鼓楼夜话】山东一男子因强奸10岁女童获刑12年，而其恶行暴露的原因，是因为女孩到医院就诊时，接诊医生意识到可能存在性侵害，按照强制报告制度要求向卫生主管部门报告。这一案例再次提醒我们，保护孩子，不只是学校和家庭的责任，医院、社区等任何有可能和孩子产生交集的部门，在特定情况下都有可...

1万赞 · 183评论 · 540万展现 · 37分钟前


 00:38

“我儿子也是当兵的！”吐鲁番大叔给驻训武警打了200个馕

视频 新华网客户端 · 229评论 · 45分钟前


英国首相后悔了

国际 澎湃新闻 · 32评论 · 52分钟前

 00:19

美国脱口秀：中国孩子已重返校园，我们美国孩子呢？

视频 北京周报 · 42评论 · 1小时前



中国日报网 未关注 · 中国日报网官方账号

【#苏伊士运河拥堵 船只或将绕道南非#】当地时间3月26日，南非航运专家科普勒在接受媒体采访时表示，鉴于苏伊士运河日前发生的搁浅货轮事故，随着延误继续，许多等待的船只必须考虑延误成本和货物的时效性要求，如果苏...

166赞 · 73评论 · 53万展现 · 1小时前

1小时前看到这里 点击刷新