21.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
}
```

1. 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
}
```

1. 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 的

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

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 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。

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。

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。

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

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. 现代 C++：一文读懂智能指针 - 知乎 (zhihu.com)
2. 详解C++11智能指针 - 孤独的猫 - 博客园 (cnblogs.com)