C++11 unique_ptr智能指针

在《C++11 shared_ptr智能指针》的基础上，本节继续讲解 C++11 标准提供的另一种智能指针，即 unique_ptr 智能指针。

作为智能指针的一种，unique_ptr 指针自然也具备“在适当时机自动释放堆内存空间”的能力。和 shared_ptr 指针最大的不同之处在于，unique_ptr 指针指向的堆内存无法同其它 unique_ptr 共享，也就是说，每个 unique_ptr 指针都独自拥有对其所指堆内存空间的所有权。

> 这也就意味着，每个 unique_ptr 指针指向的堆内存空间的引用计数，都只能为 1，一旦该 unique_ptr 指针放弃对所指堆内存空间的所有权，则该空间会被立即释放回收。

unique_ptr 智能指针是以模板类的形式提供的，unique_ptr<T>（T 为指针所指数据的类型）定义在 <memory> 头文件，并位于 std 命名空间中。因此，要想使用 unique_ptr 类型指针，程序中应首先包含如下 2 条语句：

```
01.  #include <memory>
02.  using namespace std;
```

> 第 2 句并不是必须的，可以不添加，则后续在使用 unique_ptr 指针时，必须标注 std:: 。

unique_ptr智能指针的创建

考虑到不同实际场景的需要，unique_ptr<T> 模板类提供了多个实用的构造函数，这里给读者列举了几种常用的构造 unique_ptr 智能指针的方式。

1) 通过以下 2 种方式，可以创建出空的 unique_ptr 指针：

```
01.  std::unique_ptr<int> p1();
02.  std::unique_ptr<int> p2(nullptr);
```

2) 创建 unique_ptr 指针的同时，也可以明确其指向。例如：

```
01.  std::unique_ptr<int> p3(new int);
```

由此就创建出了一个 p3 智能指针，其指向的是可容纳 1 个整数的堆存储空间。

> 和可以用 make_shared<T>() 模板函数初始化 shared_ptr 指针不同，C++11 标准中并没有为 unique_ptr 类型指针添加类似的模板函数。

3) 基于 unique_ptr 类型指针不共享各自拥有的堆内存，因此 C++11 标准中的 unique_ptr 模板类没有提供拷贝构造函数，只提供了移动构造函数。例如：

```
std::unique_ptr<int> p4(new int);
std::unique_ptr<int> p5(p4);//错误，堆内存不共享
std::unique_ptr<int> p5(std::move(p4));//正确，调用移动构造函数
```

值得一提的是，对于调用移动构造函数的 p4 和 p5 来说，p5 将获取 p4 所指堆空间的所有权，而 p4 将变成空指针（nullptr）。

4) 默认情况下，unique_ptr 指针采用 std::default_delete<T> 方法释放堆内存。当然，我们也可以自定义符合实际场景的释放规则。值得一提的是，和 shared_ptr 指针不同，为 unique_ptr 自定义释放规则，只能采用函数对象的方式。例如：

```
//自定义的释放规则
struct myDel
{
    void operator()(int *p) {
        delete p;
    }
};
std::unique_ptr<int, myDel> p6(new int);
//std::unique_ptr<int, myDel> p6(new int, myDel());
```

unique_ptr<T>模板类提供的成员方法

为了方便用户使用 unique_ptr 智能指针，unique_ptr<T> 模板类还提供有一些实用的成员方法，它们各自的功能如表 1 所示。

表 1 unique_ptr指针可调用的成员函数

成员函数名	功 能
operator*()	获取当前 unique_ptr 指针指向的数据。
operator->()	重载 -> 号，当智能指针指向的数据类型为自定义的结构体时，通过 -> 运算符可以获取其内部的指定成员。
operator =()	重载了 = 赋值号，从而可以将 nullptr 或者一个右值 unique_ptr 指针直接赋值给当前同类型的 unique_ptr 指针。

operator []()	重载了 [] 运算符，当 unique_ptr 指针指向一个数组时，可以直接通过 [] 获取指定下标位置处的数据。
get()	获取当前 unique_ptr 指针内部包含的普通指针。
get_deleter()	获取当前 unique_ptr 指针释放堆内存空间所用的规则。
operator bool()	unique_ptr 指针可直接作为 if 语句的判断条件，以判断该指针是否为空，如果为空，则为 false；反之为 true。
release()	释放当前 unique_ptr 指针对所指堆内存的所有权，但该存储空间并不会被销毁。
reset(p)	其中 p 表示一个普通指针，如果 p 为 nullptr，则当前 unique_ptr 也变成空指针；反之，则该函数会释放当前 unique_ptr 指针指向的堆内存（如果有），然后获取 p 所指堆内存的所有权（p 为 nullptr）。
swap(x)	交换当前 unique_ptr 指针和同类型的 x 指针。

> 除此之外，C++11标准还支持同类型的 unique_ptr 指针之间，以及 unique_ptr 和 nullptr 之间，做 ==，!=，<，<=，>，>= 运算。

下面程序给大家演示了 unique_ptr 智能指针的基本用法，以及该模板类提供了一些成员方法的用法：

```
#include <iostream>
#include <memory>
using namespace std;

int main()
{
    std::unique_ptr<int> p5(new int);
    *p5 = 10;
    // p 接收 p5 释放的堆内存
    int * p = p5.release();
    cout << *p << endl;
    //判断 p5 是否为空指针
    if (p5) {
        cout << "p5 is not nullptr" << endl;
    }
    else {
        cout << "p5 is nullptr" << endl;
    }

    std::unique_ptr<int> p6;
    //p6 获取 p 的所有权
```

```
22.         p6.reset(p);
23.         cout << *p6 << endl;;
24.         return 0;
25.     }
```

程序执行结果为：

```
10
p5 is nullptr
10
```