C++11 在创建 Windows 智能指针方面的威力

迈克尔·乔达基斯

2015 年 7 月 16 日 警察

评价我: 4.83/5 (15 票)

某些 Windows 句柄的智能指针

介绍

是的，你想要`shared_ptr<HANDLE>`。但这并不容易。让我们尝试创建两个类，`shared_handle`并且`dup_handle` 可以帮助我们更轻松地管理我们的Windows句柄。

问题

我们在 Windows 中使用智能指针时遇到了一些问题：

- `HANDLE`是一个`typedef`到`void*`。因此，`shared_ptr<HANDLE>`不会创建 shared `HANDLE`，它会创建一个指向 a 的共享指针`HANDLE`。
- 不同的`HANDLE`类型需要不同的复制和销毁策略。例如，`HICON`and`HCURSOR`是两个相等的类型（`typedef HICON HCURSOR`）但要使用不同的函数来销毁 an`HICON`或 a `HCURSOR`（`DestroyIcon`, `DestroyCursor`）。此外，`FindFirstFile()` 也返回 a`HANDLE`但这不能被销毁`CloseHandle()`。因此，我们的类仅适用于窗口句柄的子集。

因此，我们只能做我们能做的，但我在这里提供两种解决方案：

- `dup_handle` 将复制 a 的a`HBITMAP`或`HANDLE`可以与 复制的 a `DuplicateHandle()`。
- A`shared_handle` 将简单地复制句柄，直到最后一个`shared_handle`被销毁，在这种情况下，句柄也被销毁。

无效政策

有两种情况，`HANDLE`s 的时候无效`INVALID_HANDLE_VALUE` ，其余的时候无效`0`：

C++ 复制代码

```
// Invalid handle classes
// ----------------------------

// Everything except HANDLE
template<typename T>
class invalidation_policy
    {
```

```
    public:
        static T inv()
            {
            return 0;
            }
    };

// HANDLE
template<>
class invalidation_policy<HANDLE>
    {
    public:
        static HANDLE inv()
            {
            return INVALID_HANDLE_VALUE;
            }
    };
```

销毁政策

有两种情况：一种用于 a HGDIOBJ，被 销毁DeleteObject()，另一种用于HANDLE被 销毁CloseHandle()。不幸的是，HGDIOBJ与(grrrr)相同typedef，HANDLE所以我们必须使用一些类型特征：

C++ 缩小▲ 复制代码

```
/// GDI Objects
template <typename T>
struct is_gdiobj
    {
    static const bool value = false;
    };
template<> struct is_gdiobj<HBITMAP> { static const bool value = true; };
template<> struct is_gdiobj<HPEN> { static const bool value = true; };
template<> struct is_gdiobj<HBRUSH> { static const bool value = true; };
template<> struct is_gdiobj<HFONT> { static const bool value = true; };
template<> struct is_gdiobj<HRGN> { static const bool value = true; };
template<> struct is_gdiobj<HPALETTE> { static const bool value = true; };

// Destruction policy
// ------------------------------
// Everything Else
template<typename T,typename J = T>
class destruction_policy
    {
    public:
        static void destruct(T h)
            {
            CloseHandle(h);
            }
    };

// GDI Objects
template <typename T>
class destruction_policy<T,typename std::enable_if<is_gdiobj<T>::value,T>::type>
    {
    public:
        static void destruct(T h)
            {
            DeleteObject(h);
            }
    };
```

复制政策

我们shared_handle 也有重复，所以我们有一个可以用 复制某些句柄的情况DuplicateHandle()，我还为 提供了重复HBITMAP：

C++ 复制代码

```
// Duplicate handle classes
// ----------------------------
template<typename T>
class dup_policy
    {
    public:
        static T dup(T h)
            {
            T hY = 0;
            DuplicateHandle(GetCurrentProcess(),h,GetCurrentProcess(),
                                &hY,0,true,DUPLICATE_SAME_ACCESS);
            return hY;
            }
    };

// HBITMAP
template<>
class dup_policy<HBITMAP>
    {
    public:
        static HBITMAP dup(HBITMAP h)
            {
            HBITMAP hY = (HBITMAP)CopyImage(h,IMAGE_BITMAP,0,0,0);
            return hY;
            }
    };
```

dup_handle

该dup_handle 绝：

- 从 Windows 句柄构造。
- 复制语义：关闭自身，复制句柄。
- 移动语义：关闭自身，移动手柄，使目标无效。
- 破坏：破坏手柄

共享句柄

该shared_handle绝：

- 从 Windows 句柄构造。
- 复制语义：如果唯一，则关闭自身，复制原始句柄。
- 移动语义：如果唯一，则关闭自身，复制原始句柄。
- 销毁：如果唯一则关闭自身

shared_handle 在std::shared_ptr 内部使用 a来测试句柄是否唯一。

完整代码

因此，整个代码shared_handle 和dup_handle 看起来像：

C++ 缩小▲ 复制代码

```
#include <windows.h>
#include <memory>
#include <type_traits>
```

```
namespace WINPTR
    {
    // Specializations for specific types

    /// GDI Objects
    template <typename T>
    struct is_gdiobj
        {
        static const bool value = false;
        };
    template<> struct is_gdiobj<HBITMAP> { static const bool value = true; };
    template<> struct is_gdiobj<HPEN> { static const bool value = true; };
    template<> struct is_gdiobj<HBRUSH> { static const bool value = true; };
    template<> struct is_gdiobj<HFONT> { static const bool value = true; };
    template<> struct is_gdiobj<HRGN> { static const bool value = true; };
    template<> struct is_gdiobj<HPALETTE> { static const bool value = true; };

    // Invalid handle classes
    // ----------------------------
    // Everything except HANDLE
    template<typename T>
    class invalidation_policy
        {
        public:
            static T inv()
                {
                return 0;
                }
        };

    // HANDLE
    template<>
    class invalidation_policy<HANDLE>
        {
        public:
            static HANDLE inv()
                {
                return INVALID_HANDLE_VALUE;
                }
        };

    // Destruction policy
    // ----------------------------
    // Everything Else
    template<typename T,typename J = T>
    class destruction_policy
        {
        public:
            static void destruct(T h)
                {
                CloseHandle(h);
                }
        };

    // GDI Objects
    template <typename T>
    class destruction_policy<T,typename std::enable_if<is_gdiobj<T>::value,T>::type>
        {
        public:
            static void destruct(T h)
                {
                DeleteObject(h);
                }
        };

    // Duplicate Policy
    // ----------------------------
    template<typename T>
```

```
class dup_policy
    {
    public:
        static T dup(T h)
            {
            T hY = 0;
            DuplicateHandle(GetCurrentProcess(),h,GetCurrentProcess(),
                            &hY,0,true,DUPLICATE_SAME_ACCESS);
            return hY;
            }
    };

// HBITMAP
template<>
class dup_policy<HBITMAP>
    {
    public:
        static HBITMAP dup(HBITMAP h)
            {
            HBITMAP hY = (HBITMAP)CopyImage(h,IMAGE_BITMAP,0,0,0);
            return hY;
            }
    };

template <typename T = HANDLE,typename Destruction = destruction_policy<T>,
  typename Invalidation = invalidation_policy<T>,typename Duplication = dup_policy<T>>
class dup_handle
    {
    private:
        T hX = Invalidation::inv();

    public:

        dup_handle()
            {
            hX = Invalidation::inv();
            }
        ~dup_handle()
            {
            Close();
            }
        dup_handle(const dup_handle& h)
            {
            hX = Duplication::dup(h.hX);
            }
        dup_handle(dup_handle&& h)
            {
            Move(std::forward<dup_handle>(h));
            }
        dup_handle(T hY)
            {
            hX = hY;
            }
        dup_handle& operator =(const dup_handle& h)
            {
            Close();
            hX = Duplication::dup(h.hX);
            return *this;
            }
        dup_handle& operator =(dup_handle&& h)
            {
            Move(std::forward<dup_handle>(h));
            return *this;
            }

        void Close()
            {
            if (hX != Invalidation::inv())
                Destruction::destruct(hX);
```

```
            hX = Invalidation::inv();
            }

        void Move(dup_handle&& h)
            {
            Close();
            hX = h.hX;
            h.hX = Invalidation::inv();
            }
        operator T() const
            {
            return hX;
            }
    };

template <typename T = HANDLE,typename Destruction = destruction_policy<T>,
                      typename Invalidation = invalidation_policy<T>>
class shared_handle
    {
    private:
        T hX = Invalidation::inv();
        std::shared_ptr<size_t> ptr = std::make_shared<size_t>();

    public:

        // Closing items
        void Close()
            {
            if (!ptr || !ptr.unique())
                {
                ptr.reset();
                return;
                }
            ptr.reset();
            if (hX != Invalidation::inv())
                Destruction::destruct(hX);
            hX = Invalidation::inv();
            }

        shared_handle()
            {
            hX = Inv();
            }
        ~shared_handle()
            {
            Close();
            }
        shared_handle(const shared_handle& h)
            {
            Dup(h);
            }
        shared_handle(shared_handle&& h)
            {
            Move(std::forward<shared_handle>(h));
            }
        shared_handle(T hY)
            {
            hX = hY;
            }
        shared_handle& operator =(const shared_handle& h)
            {
            Dup(h);
            return *this;
            }
        shared_handle& operator =(shared_handle&& h)
            {
            Move(std::forward<shared_handle>(h));
            return *this;
            }
```

```
            void Dup(const shared_handle& h)
                {
                Close();
                hX = h.hX;
                ptr = h.ptr;
                }
            void Move(shared_handle&& h)
                {
                Close();
                hX = h.hX;
                ptr = h.ptr;
                h.ptr.reset();
                h.hX = Inv();
                }
            operator T() const
                {
                return hX;
                }
        };
    }
```

测试

C++ 复制代码

```
int main()
    {
    using namespace WINPTR;
    dup_handle<> hX = OpenProcess(...);
    shared_handle<> hY = GetCurrentThread();

    dup_handle<> z1 = hX;
    shared_handle<HBITMAP> b2 = LoadBitmap(...);

    auto cc = hY;
    return 0; // Auto destroy for all handles
    }
```

历史

- 16/07/2015：第一个想法

执照

分享

关于作者

迈克尔·乔达基斯
软件开发人员
希腊
手表该会员

我正在使用 C++、PHP、Java、Windows、iOS、Android 和 Web（HTML/Javascript/CSS）。

我拥有数字信号处理和人工智能博士学位，专攻专业音频和人工智能应用。

我的主页：https://www.turbo-play.com

评论和讨论

添加评论或问题 | 电子邮件提醒 | Search Comments

第一 页上一页 下一页

我的5票
Farhad Reza 8-Oct-15 20:48

一个小小的并发症
peterchen 21-Jul-15 22:45

为什么不专门研究 shared_ptr?
Brian J Rothwell 18-Jul-15 7:01

Re: 为什么不专门研究 shared_ptr?
Michael Chourdakis 18-Jul-15 7:05

Re: 为什么不专门研究 shared_ptr?
peterchen 21-Jul-15 22:33

Re: 为什么不专门研究 shared_ptr?
Michael Chourdakis 21-Jul-15 22:36

Re: 为什么不专门研究 shared_ptr?
peterchen 21-Jul-15 22:53

Re: 为什么不专门研究 shared_ptr?
Michael Chourdakis 21-Jul-15 23:49

Re: 为什么不专门研究 shared_ptr?
peterchen 22-Jul-15 17:14

Re: 为什么不专门研究 shared_ptr?
Michael Chourdakis 27-Aug-15 23:47

刷新 1

一般 新闻 建议 问题 错误 答案 笑话 赞美 咆哮 管理员

使用Ctrl+Left/Right 切换消息，Ctrl+Up/Down 切换主题，Ctrl+Shift+Left/Right 切换页面。

永久链接
广告
隐私
Cookie
使用条款

布局：固定 | 体液