0. 课堂内容复习演练

在 godbolt.org 上观察 move 之后对象的状态，思考“原对象处于有效状态”的语义。

```
#include <string>
#include <iostream>

int main()
{
    std::string str0{"hello"};

    std::string&& str1 = std::move(str0);
    str0 += '!';
    std::cout << str0 << std::endl;
    std::cout << str1 << std::endl;

    std::cout << "------" << std::endl;
    std::string str2(str1);
    std::cout << str0 << std::endl;
    std::cout << str1 << std::endl;
    std::cout << str2 << std::endl;

    std::cout << "------" << std::endl;
    std::string str3(std::move(str1));
    std::cout << str0 << " is " << str0.empty() << std::endl;
    std::cout << str1 << " is " << str1.empty() << std::endl;
    std::cout << str2 << std::endl;
    std::cout << str3 << std::endl;
}
```

godbolt
仍有效的一个例子

```
#include <vector>
#include <iostream>

int main()
{
    std::vector<int> v = {0,1,2,3,4};
    std::vector<int>&& refv = std::move(v);
    std::vector<int> v1 = std::move(v);

    auto dump = [](auto&& vec){
        std::cout << "---" << std::endl;
        for(auto& i: vec)
        {
            std::cout << i << std::endl;
        }
        std::cout << "===" << std::endl;
    };

   // dump(v);
```

```
    dump(v1);
    std::cout << v1.empty() << std::endl;
}
```

godbolt

右值引用判别。利用自 C++11 引入的 RTTI 机制在运行期检测表达式的左右值属性，尝试分析以下程序的打印输出:

```
#include <iostream>
#include <type_traits>
#include <string>
#include <iostream>

int main()
{
    int x;
    int& y = x;
    int&& z = std::move(x);
    auto&& f = []{};

    std::cout << std::is_reference<decltype(x)>::value << std::endl
              << std::is_reference<decltype((x))>::value << std::endl
              << std::is_reference<decltype(y)>::value << std::endl
              << std::is_reference<decltype((y))>::value << std::endl
              << std::is_reference<decltype(z)>::value << std::endl
              << std::is_reference<decltype((z))>::value << std::endl
              << std::is_reference<decltype(f)>::value << std::endl
              << std::is_reference<decltype((f))>::value << std::endl;

    std::cout << "-------------" << std::endl;

    std::cout << std::is_rvalue_reference<decltype(x)>::value << std::endl
              << std::is_rvalue_reference<decltype((x))>::value << std::endl
              << std::is_rvalue_reference<decltype(y)>::value << std::endl
              << std::is_rvalue_reference<decltype((y))>::value << std::endl
              << std::is_rvalue_reference<decltype(z)>::value << std::endl
              << std::is_rvalue_reference<decltype((z))>::value << std::endl
              << std::is_rvalue_reference<decltype(f)>::value << std::endl
              << std::is_rvalue_reference<decltype((f))>::value << std::endl;
}
```

godbolt

移动语义的工业级应用

资源所有权，用到 `std::vector` 标准等容器的的地方常会伴随着 `std::move` 的使用：
first party set, web cookie
browser view

函数对象声明周期控制，callback 机制，目前关注语义和使用。

> base::OnceCallback<> is created by base::BindOnce(). This is a callback variant that is a move-only type and can be run only once.

```
// ...;
  using LoadCompleteOnceCallback = base::OnceCallback<void(
      base::flat_map<net::SchemefulSite, net::SchemefulSite>)>;
// ...;
// We use a OnceCallback to ensure we only pass along the completed sets once.
  LoadCompleteOnceCallback on_load_complete_
      GUARDED_BY_CONTEXT(sequence_checker_);
// ...;

void FirstPartySetsLoader::MaybeFinishLoading() {
  DCHECK_CALLED_ON_VALID_SEQUENCE(sequence_checker_);
  if (!HasAllInputs())
    return;
  ApplyManuallySpecifiedSet();
  ApplyAllPolicyOverrides();
  std::move(on_load_complete_).Run(std::move(sets_));
}

  R Run(Args... args) && {
    // Move the callback instance into a local variable before the invocation,
    // that ensures the internal state is cleared after the invocation.
    // It's not safe to touch |this| after the invocation, since running the
    // bound function may destroy |this|.
    OnceCallback cb = std::move(*this);
    PolymorphicInvoke f =
        reinterpret_cast<PolymorphicInvoke>(cb.polymorphic_invoke());
    return f(cb.bind_state_.get(), std::forward<Args>(args)...);
  }
```

[chromium call back mechanism doc]
[chromium call back header]

Tensorflow 类似的使用：

资源容器：
[MakeOverviewPageTip]
函数对象声明周期的控制
[host stream]

1. RAII

ScopedGuard

```
template<typename Func>
class ScopeGuard final
{
   public:
      ScopeGuard(Func func) : m_Func(func) {}
      ~ScopeGuard() { m_Func(); }
   private:
      Func m_Func;
};
void SomeFunc()
{
   FILE* fp = fopen("filename", "wb");
   ScopeGuard<std::function<void()>> guard([&fp]() { fclose(fp); });
   //Process
}
```

计时器

参照以下代码实现一个 RAII 计时器，并用它直观比较 move 语义带来的速度提升。

```
#include <chrono>
#include <iostream>
#include <vector>

class RaiiTimer {
   public:
      RaiiTimer()
      {
         std::cout << "-------" << std::endl;
         b = std::chrono::steady_clock::now();
      }
      ~RaiiTimer()
      {
         std::cout << std::chrono::duration<double>
            ( std::chrono::steady_clock::now() - b).count()
            << std::endl;
      }
   private:
      std::chrono::time_point<std::chrono::steady_clock> b;

};

int main()
{
   std::vector<std::string> v1;
   std::string x = "ABC123";
   for(int i = 0; i < 1000000; ++i)
   {
      v1.push_back(x);
```

```
    }

    {
        std::vector<std::string> v2;
        RaiiTimer t1;
        v2 = v1;
    }

    {
        std::vector<std::string> v3;
        RaiiTimer t2;
        v3 = std::move(v1);
    }
```

godbolt

工业级应用

Google chrome 中，在注释中明确指出的有近 500 处。
suppress_keyboard_raii
scoped_generic
simple_file_tracker

2. 实现移动语义

基于第一节课三法则的例子，为 CDataBuffer 提供移动语义支持，并利用上节构建的计时器比较移动语义是否带来明显的速度提升。

声明：

```
int trickster()
{
   throw std::runtime_error("error");
}

class CDataBuffer
 {
 public:
     CDataBuffer(const std::string& name, A* data, unsigned int length);
     CDataBuffer(const CDataBuffer& db);
     CDataBuffer& operator= (const CDataBuffer& db);
     CDataBuffer(CDataBuffer&& db) noexcept;
     CDataBuffer& operator= (CDataBuffer&& db) noexcept;
     virtual ~CDataBuffer();

     A* GetExtraData();
     void Dump ();
 private:
     std::string m_dataName;
     unsigned int m_DataLength;
     unsigned int m_BufSize = 0;
     A* m_pFoo;
 }
```

在写好的 CDataBuffer 上观察返回值优化对副作用的忽略，思考这对 RAII 可能带来的影响。

进阶要求

利用 copy swap 惯用法实现，并利用 trickster 函数实验增加异常安全测试

工业级实现

Qt qxmlstream
Qcursor
Qpalette
image in chromium