Lecture 4

TOC

左值，右值

C++11 引入了右值引用的概念，使得我们可以更好地理解对象的生命周期。

- 广义左值（glvalue）：可以取地址的表达式，在赋值运算中位于等号左侧
  - 左值(lvalue): 可以取地址、持久存在的对象。
  - 亡值(xvalue): 即将销毁，资源可以被重用的对象。
- 右值：不能持有持久存储位置的表达式，它们通常是临时对象或常量，不能出现在赋值操作的左边。右值是表达式的结果或者是常量值。
  - 纯右值(prvalue)：无法取地址的临时对象或常量，比如字面量、临时对象等。
  - 亡值(xvalue)

Reference

移动语义 `std::move()`

`std::move()` 返回指向当前左值的右值引用，用于将左值转换为右值，从而实现资源的转移。这在类的构造中非常有用，能减少资源拷贝的开销。

```
int a = 10;
int&& r_ref = std::move(a);
```

引用坍缩和完美转发

- 模板元编程中常用，这里不过多展开。感兴趣可以自行学习。

Link

别名(Alias)

C 语言中，使用 `typedef` 关键字定义别名。C++11 引入了 `using` 关键字，提供了更多对模板的支持。

```cpp
#include <cstdint>

typedef int32_t i32;
using u8 = uint8_t;

typedef void (*Func)(int);

template <typename T1, typename T2> struct foo {
  T1 a;
  T2 b;
};

typedef foo<int, double> TypeDefFoo;

template <typename T1> using Using = foo<T1, int>;
```

auto 自动类型推导

C++11 改变了 auto 关键字的用法，使得 C++ 能够像其他新兴语言一样进行类型推导。

```
auto a = 1; // int
auto b = 1.0; // double
auto c = "hello"; // const char*
```

部分应用：

- Range-based for

```
int arr[] = {1, 2, 3, 4, 5};
for (auto i : arr) {
  std::cout << i << '\n';
}
```

- 新的函数定义写法

```
auto foo() → int {
  return 1;
}
```

- Universal Reference

使用 `auto&&` 可以实现完美转发，延迟类型推导，避免不必要的拷贝。

```
auto&& res = foo();
```

面向对象编程

C 语言是一种面向过程的编程语言，也就是常说的

Q&A