第6章 右值引用

《现代C++语言核心特性解析》 谢丙堃

区分左值和右值

- 不用左右区分

```
int a = 1;
int b = a;
```

- 通常的所说左值和右值的判断方法
  - 判断对象能否取地址
  - 所谓的左值一般是指一个指向特定内存具有名称的值（具名对象），它有一个相对稳定内存地址，并且有用一段较长的生命周期。而右值则是不指向稳定内存地址匿名值（不具名对象），它的生命周期很短，通常都是暂时性的。

左值引用和右值引用

- 常量左值引用

```
int &x1 = 7;          // 编译错误
const int &x = 11;    // 编译成功
```

- 右值引用语法

```
int &&k = 11;         // 右值引用
```

使用右值引用

```
class X {
public:
    X() { std::cout << "X ctor" << std::endl; }
    X(const X&x) { std::cout << "X copy ctor" << std::endl; }
    ~X() { std::cout << "X dtor" << std::endl; }
    void show() { std::cout << "show X" << std::endl; }
};
X make_x() {
    X x1;
    return x1;
}
int main() {
    X &&x2 = make_x();
    x2.show();
}
```

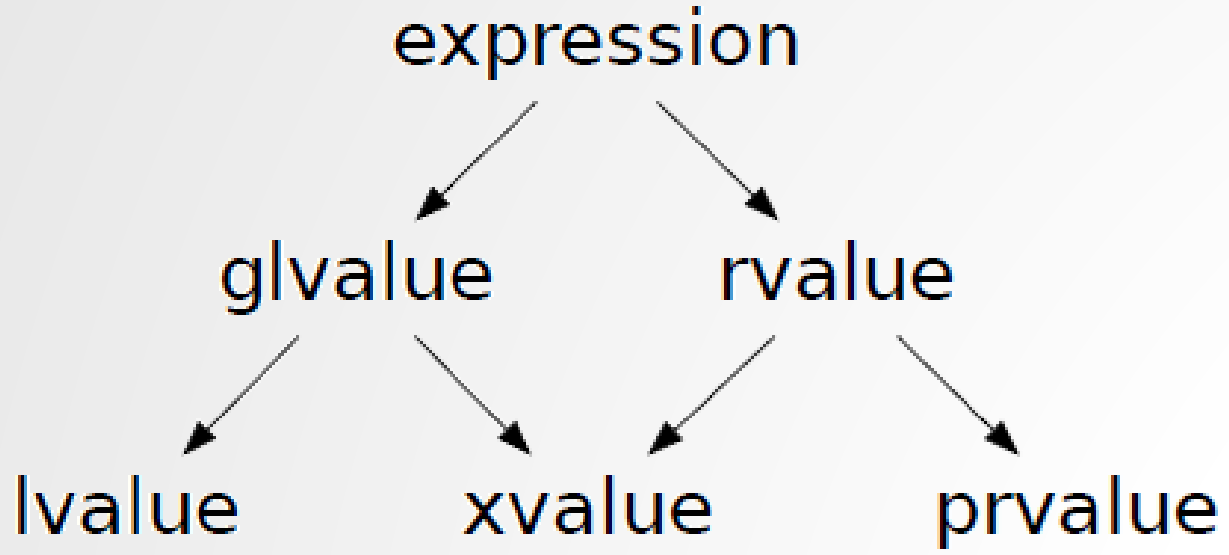
移动语义

- 自动生成的移动构造函数
- 移动的风险

```
class BigMemoryPool {
public:
    BigMemoryPool(BigMemoryPool&& other)
    {
        pool_ = other.pool_;
        other.pool_ = nullptr;
    }
...
private:
    char *pool_;
};
```

值类别

- 泛左值（glvalue)和右值（rvalue）
- 左值、纯右值和将亡值



将左值转换为右值

- 基本的转换方法

```
int i = 0;
int &&k1 = i;   // 编译失败
int &&k2 = static_cast<int&&>(i);   // 编译成功
```

- 使用std::move
  - 等价于以下代码

```
static_cast<std::remove_reference<decltype(arg)>::type&&>(arg)
```

万能引用和引用折叠

- 基本语法

```
void foo(int &&i) {}     // i为右值引用

template<class T>
void bar(T &&t) {}       // t为万能引用
```

- 引用折叠规则

类模板型	T实际类型	最终类型
T&	R	R&
T&	R&	R&
T&	R&&	R&
T&&	R	R&&
T&&	R&	R&
T&&	R&&	R&&

完美转发

- 对比

```cpp
template<class T>
void normal_forwarding1(T t)
{
    show_type(t);
}

template<class T>
void normal_forwarding2(T& t)
{
    show_type(t);
}

template<class T>
void normal_forwarding3(const T& t)
{
    show_type(t);
}
```

```cpp
template<class T>
void perfect_forwarding(T &&t)
{
    show_type(static_cast<T&&>(t));
}
```

使用 std::forward

感谢聆听
欢迎关注