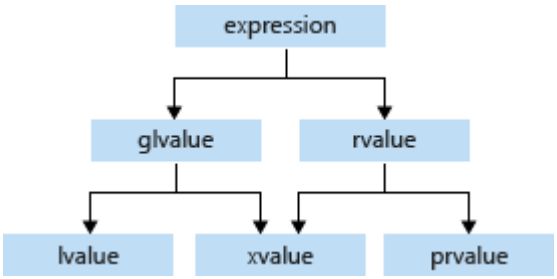
Lvalues 和 Rvalues (C++)

每个 C++ 表达式都有一个类型，属于值类别。 值类别是编译器在表达式计算期间创建、复制和移动临时对象时必须遵循的规则的基础。

C++17 标准对表达式值类别的定义如下：

- glvalue 是一个表达式，它的计算可以确定对象、位域或函数的标识。
- prvalue 是一个表达式，它的计算可以初始化对象或位域，或计算运算符的操作数值，这是由它出现的上下文所指定的。
- xvalue 是一个 glvalue，表示一个对象或位域，该对象或位域的资源可重复使用（通常是因为它接近其生存期的末尾）。 示例：某些涉及 rvalue 引用 (8.3.2) 的类型的表达式会生成 xvalue，例如对返回类型为 rvalue 引用或强制转换为 rvalue 引用类型的函数的调用。
- lvalue 为非 xvalue 的 glvalue。
- rvalue 是一个 prvalue 或 xvalue。

下图阐释了各类别之间的关系：



该图以带框标记的表达式开头，其有两个子项：glvalue 和 rvalue。

- glvalue 有两个子项：lvalue 和 xvalue。
- rvalue 有两个子项：prvalue 和 xvalue；
- xvalue 是rvalue的子项，也是 glvalue 的子项。

---

lvalue 具有程序可访问的地址。 例如，lvalue 表达式包括变量名称，其中包括 `const` 变量、数组元素、返回 lvalue 引用的函数调用、位域、联合和类成员。

prvalue 表达式没有可供程序访问的地址。 例如，prvalue 表达式包括文本、可返回非引用类型的函数调用，以及在表达式计算期间创建的但只能由编译器访问的临时对象。

xvalue 表达式有一个地址，该地址不再可供程序访问，但可用于初始化 rvalue 引用，以提供对表达式的访问。 例如，它包括可返回 rvalue 引用的函数调用，以及数组下标、成员和指向其中数组或对象是 rvalue 引用的成员表达式的指针。

这段描述主要涉及 C++ 中的左值（lvalue）和右值（rvalue）概念，以及通过 xvalue 表达式提供对右值的访问。

- **左值（lvalue）**：指向内存位置的表达式，对其进行赋值操作是合法的，并且可以取地址。例如，变量和通过引用访问的对象都是左值。

- **右值（rvalue）**：不存储在内存中的临时值，对其进行赋值操作是不合法的，并且通常是一次性的。例如，常量、临时变量和返回临时对象的函数调用结果都是右值。

xvalue (expiring value) 是C++11中引入的一种新类型的值，是右值的一种特殊形式。xvalue 表达式可以是将要销毁的对象的引用，但仍然允许使用该对象的值。它通常出现在移动语义相关的操作中。

关于描述中提到的情况：

1. **可返回 rvalue 引用的函数调用**：这意味着函数返回一个右值引用（通常是通过移动语义实现），允许将其返回值视为右值，而不会产生额外的拷贝。例如：

```
cppCopy codeint&& foo() {
    int x = 42;
    return std::move(x); // x 被视为右值，可以通过移动语义返回
}

int main() {
    int&& value = foo(); // 使用右值引用初始化
    return 0;
}
```

2. **数组下标、成员和指向其中数组或对象是 rvalue 引用的成员表达式的指针**：这意味着通过指针访问的表达式是右值引用，例如：

```
cppCopy code#include <iostream>

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int* ptr = &arr[0]; // 指向数组的第一个元素

    int&& value = std::move(*ptr); // 使用指针访问数组元素，返回右值引用

    std::cout << value << std::endl;

    return 0;
}
```

在这两种情况下，通过 xvalue 表达式，可以将右值引用绑定到即将销毁的对象上，以便进行进一步的操作，而无需额外的拷贝或创建新的左值对象。

示例

以下示例演示左值和右值的多种正确的和错误的用法：

```
// lvalues_and_rvalues2.cpp
int main()
{
    int i, j, *p;

    // Correct usage: the variable i is an lvalue and the literal 7 is a
prvalue.
```

```
    i = 7;

    // Incorrect usage: The left operand must be an lvalue (C2106).`j * 4` is a
prvalue.
    7 = i; // C2106
    j * 4 = 7; // C2106

    // Correct usage: the dereferenced pointer is an lvalue.
    *p = i;

    // Correct usage: the conditional operator returns an lvalue.
    ((i < 3) ? i : j) = 7;

    // Incorrect usage: the constant ci is a non-modifiable lvalue (C3892).
    const int ci = 7;
    ci = 9; // C3892
}
```

> 此主题中的示例阐释了未重载运算符时的正确和错误用法。 通过重载运算符，可以使表达式（如 `j * 4` ）成为左值。