右值引用和成员函数

拷贝构造和拷贝赋值参数一般为const Foo &，因为它不需要改变源对象，并且const Foo &可以绑定到任何对象上，常量左值，非常量左值，右值等。

移动构造和移动赋值，一般会改变源对象（移动操作无法保证移动后源对象的值是什么，只需保证该对象是个有效状态即可），一般是Foo &&，不需要const Foo &&。

左值和右值引用成员函数

通常，不论是左值对象还是右值对象都是可以调用成员函数的。

```
string s1="123",s2="456";
auto n=(s1+s2).find('a');    // 右值调用成员find
```

此外，旧标准中右值也可以被赋值（虽然没什么意义），

```
s1+s2="wow";
```

新标准为了维持向后兼容，允许向右值赋值。但是我们希望在自己定义的类中阻止这种用法。**此情况下，我们希望强制左侧运算对象是一个左值。**

这种方式和const成员函数修饰this一样，即在参数列表后放置一个**引用限定符**:

> 这里的&和&&也是修饰this的，指明this可以绑定左值还是右值。

```
class Foo{
    public:
        Foo &operator=(const Foo &) &;     // 只能向可修改的左值赋值
};
Foo &Foo::operator=(const Foo &rhs) &{
    // 执行rhs赋给本对象的工作
    return *this;
}
```

> 类似const，引用限定符只能作用于nonstatic成员函数，且必须同时出现在函数的声明和定义中。

正确用法如下，const应在&,&&之前。

```
class Foo{
Foo fuc()const &;
Foo fuc()const &&;
};
```

重载和引用函数

> 成员函数可以通过有无const来区分。这里可以引申一下，可以查看“函数匹配和const.md”。

引用限定符也可以区分重载版本。而且，我们可以综合const和引用限定符来区分一个成员函数的重载版本。例如，

```
class Foo{
public:
    Foo sorted() &&;                // 可用于可修改的右值，即this指针可以绑定到nonconst右值上
    Foo sorted() const &;           // 可用于任何类型的Foo，左值，右值，常量左值，常量右值，都可以
                                    // 即如果没有上面一个版本，右值对象调用sorted，将会调用此版本
    // Foo sorted() &; 此版本的sorted可以区分左值和右值调用，但是不可以调用常量左值和右值
private:
    vector<int> data;
};
// 本对象为右值，因此可以原址排序
Foo Foo::sorted()&&{
    sort(data.begin(),data.end());
    return *this;
}
// 本对象是const或是一个左值，不可以原址排序
Foo Foo::sorted()const &{
    Foo ret(*this);
    sort(ret.data.begin(),ret.data.end());
    return ret;
}
```

编译器会根据调用sorted的对象的左值/右值属性来确定使用哪一个sorted版本：

```
Rvalue_objet.sorted();      // Rvalue_object是一个非const右值，Foo::sorted()&&是最佳匹配，即Foo::sorted()const &也是一个匹配候选函数
Lvalue_object.sorted();     // 调用Foo::sorted()const &，Foo::sorted()&&不是候选函数
```

> 我们定义const成员函数时，可以定义两个版本，唯一的差别就是一个版本有const限定而另一个没有。

当我们定义两个或以上的同名且同参数的成员函数，必须对所有函数都加上引用限定符或者所有都不加

```
class Foo{
public:
    Foo sorted()&&;
    Foo sorted()const ;// 错误，这将造成匹配二义性，该版本和Foo sorted()&&，都是一样好的匹配，对于右值而言，因此不允许这样的定义，必须都加上引用限定符或不加
    // Foo sorted();    // 错误，理由类似上面
    using cmp=bool(const int &,const int &);
    // 下面两个可以正确区分
    Foo sorted(cmp*);
```

```
        Foo sorted(cmp *)const;
    }
```