|  |  |
| --- | --- |
| 1. 右值引用和成员函数 2. 右值和左值引用成员函数 3. 重载和引用函数 | Void push\_back(const x&);  VOID push\_back(X&&);  Alloc.construct(first\_Free++,std::move(s));  (s1 + s2).find(‘a’);//用右值调用函数  S1+s2 = “wow!”;对右值进行赋值  Foo & Foo::operator=(const Foo&)&  Foo someMem() const &;  Foo someMem() &&;  引用限定符 |
| 1. 右值引用和成员函数 2. 可以为成员函数提供拷贝和移动版本，能从中受益。参数是与拷贝/移动赋值运算符相同的参数模式 3. 一个是const左值引用 const X&   一个是指向非const的右值引用 X&&右值和左值引用成员函数   1. 一般来说不需要定义相对的X&和const X&&   const X&&:我们希望从实参“窃取”数据，通常是右值引用，不是const右值引用  X&： 从一个对象进行拷贝的操作通常不改变该对象   1. Alloc.construct(first\_Free++,std::move(s));,move返回右值引用，所以会使用移动构造函数来构造新元素 2. 实参类型决定新元素是拷贝还是移动到容器中。 3. 右值和左值引用成员函数 4. 右值和左值 对象都可以调用成员函数 5. 不希望右值调用或者被赋值，我们可以阻止，强制左侧运算对象（this指向的对象）是一个左值才可以调用成员函数 6. 定义方式：在成员函数参数列表后放置一个引用限定符，与定义const相同 7. 只允许可修改的左值赋值需要在拷贝赋值运算符 添加& 限定，声明与定义地方都需要 8. 与const组合使用需要放在const后   自己总结：   1. 若都对成员函数没有限定符，则this可以表示左值或右值都可以，即可以为左值或右值赋值 2. 若只对左值限定（拷贝赋值运算符），则只能对左值赋值，右值不可以 3. 若只对右值限定 则相反 4. 若对左值和右值都限定则都可以 5. 重载和引用函数 6. 引用限定符也可以区分重载版本，可以通过与const的结合重载 7. 有四种const & 、&&、&、const&&限定符但只需要   Const &：& ,const &&,const &都会转换为这个  &&,：只有&&精确匹配   1. 右值版本的函数可以直接使用，因为没有其它用户。Const&版本不能改变对象，所以需要拷贝后操作。 2. 定义const成员函数时可以有两个版本，但引用限定不行，若定义两个或两个以上具有相同名字和相同参数列表的成员函数，就必须对所有函数都加上引用限定符，或者所有都不加   代码：  Foo136 f1; // 调用 &  f1.someMem();  const Foo136 f2;  std::move(f2).someMem();// 调用const&&  void someMem() &&;  void someMem() const &;  void someMem() &;  void someMem() const &&;  void Foo136::someMem() &&  {  cout << "Foo136 Foo136::someMem() &&" << endl;  }  void Foo136::someMem() const &  {  cout << "Foo136 Foo136::someMem() const &" << endl;  }  void Foo136::someMem() &  {  cout << "Foo136 Foo136::someMem() &" << endl;  }  void Foo136::someMem() const &&  {  cout << "Foo136 Foo136::someMem() const &&" << endl;  } | |