|  |  |
| --- | --- |
| 1. 定义行为像指针的类 2. 引用计数 3. 定义一个使用引用计数的类 4. 类指针的拷贝成员 篡改 引用计数 | class HasPtr {  public:  HasPtr(const string &s = string()) :  pss(new string(s)), i(0),use(new size\_t(1)) {}  HasPtr(const HasPtr& hp):pss(hp.pss),i(hp.i),use(hp.use){  ++\*use;  }  HasPtr& operator=(const HasPtr&);  ~HasPtr() {  if (--\*use == 0) {  delete use;  delete pss;  }  }  private:  std::string \*pss;  int i;  std::size\_t\* use;  };  HasPtr& HasPtr::operator=(const HasPtr& rhs)  {  ++(\*rhs.use);  if (--\*use == 0) {  delete use;  delete pss;  }  i = rhs.i;  use = rhs.use;  pss = rhs.pss;  return \*this;  } |
| 1. 定义行为像指针的类 2. 需要为其定义拷贝构造函数和拷贝赋值运算符，拷贝指针成员本身而不是它指向的string 3. 需要自定义析构函数来释放接受string的内存，但是在本例中像指针的类中，需要判断最后一个指向string的类销毁时它才可以释放string 4. 让一个类像指针可以用shared\_ptr来管理类中的指针资源，但是我们希望自己直接管理资源，可以使用引用计数，也是动态分配的资源。 5. 引用计数 6. 引用计数，记录有多少个对象与正在创建的对象共享状态，创建一个对象时初始化为1 7. 拷贝构造函数不分配新的计数器，而是拷贝给定对象的数据成员，包括计数器。递增共享的计数器 8. 析构函数，递减计数器，若计数器为0，析构函数释放状态 9. 拷贝赋值运算符，递增右侧运算符对象的计数器，递减左侧运算对象的计数器。   需要将计数器保存在动态内存中。拷贝或赋值对象时，我们拷贝指向计数器的指针，这样副本和原对象都会指向相同的计数器。  HasPtr p1(“hi”);  HasPtr p2(p1);  HasPtr p3(p1);   1. 定义一个使用引用计数的类 2. 如上代码 3. 类指针的拷贝成员 篡改 引用计数 4. Ps(s),user(user){++\*use;} 5. 析构函数判断是否计数为0来删除 6. 拷贝赋值运算符需要执行拷贝构造函数与析构函数的工作，递增右侧，递减左侧 7. 拷贝赋值运算符还需要处理自赋值。先递增rhs中的计数，再递减左侧运算对象的，因为这两个总要执行这个操作，需按照这个顺序就没问题，这样就保证自赋值没问题 | |