|  |  |
| --- | --- |
| 1. 动态内存 2. 动态内存与智能指针 3. Shared\_ptr类 4. Make\_shared函数 5. Shared\_ptr的拷贝和赋值 6. Shared\_ptr自动销毁所管理的对象 7. Shared\_ptr还会自动释放相关联的内存 8. 使用了动态生存期的资源的类 9. 定义StrBlob类   10. StrBlob构造函数  11. 元素访问成员函数  12. StrBlob的拷贝、赋值和销毁 | 自由空间  堆  动态分配  New  Delete  智能指针  引用计数  Shared\_ptr  Unique\_ptr  Weak\_ptr  Shared\_ptr<string> p1;  Shared\_ptr<list<int>> p2;  Shared\_ptr<int> p3 = make\_shared<int>(42);  Shared\_ptr<string> p4 = make\_shared<string>(10, ‘9’);  Auto p6 = make\_shared<vector<string>>();  Auto p = make\_shared<int>(42);  Auto q(p);  析构函数 |
| 1. 动态内存 2. 静态内存或栈内存。静态内存存储static的对象数据成员，栈内存用来保存非static成员。 3. 分配在静态内存或栈内存中的对象由编译器自动创建和销毁。 4. 栈对象仅在程序块运行时才存在，static对象再使用之前分配，程序结束时销毁 5. 还有内存池。这部分内存被称为自由空间或堆。堆存储动态分配的对象，在程序运行时分配的对象。动态对象的生成期由程序控制，所以需要用代码显示销毁。   Note：动态内存使用是有必要的，但是正确的管理棘手的   1. 动态内存与智能指针 2. 动态内存的管理通过一对运算符完成：new和delete。 3. New在动态内存中为对象分配空间并返回一个指向该对象的指针 4. Delete接受一个动态对象的指针，销毁该对象释放与之关联的内存 5. 但是管理困难，会忘记销毁，会造成引用非法内存的指针，所以需要用标准库提供的两种智能指针。智能指针与常规指针类似，区别于自动销毁负责的对象。而这两种智能指针的区别又在于管理底层指针的方式。 6. Shared\_ptr允许多个指针指向同一个对象；unique\_ptr则独占所指向的对象。 7. 还有一个weak\_ptr伴随类，弱引用，指向shared\_ptr所管理的对象。在memory头文件 8. Shared\_ptr类 9. 是模板，需提供指向的类型，在尖括号中 10. 默认初始化为一个空指针 11. 使用与普通指针类似。 12. Make\_shared函数 13. 在动态内存中分配一个对象并初始化它，返回指向此对象的shared\_ptr。 14. 类似容器的emplace成员，可以用参数来构造给定类型的对象。 15. Auto更简单的创建shared\_ptr 16. Shared\_ptr的拷贝和赋值 17. 拷贝或赋值时，每个shared\_ptr都会基类有多少个其它shared\_ptr指向相同的对象 18. 有个关联的计数器，引用计数。拷贝或传参，返回值，赋值右侧，计数器会递增。赋值左侧或销毁时（离开作用域）计数器递减   Note：标准库实现记录，关键是它能在恰当时候自动释放对象   1. Shared\_ptr自动销毁所管理的对象 2. 当最后一个shared\_ptr对象被销毁时，shared\_ptr类会自动销毁此对象。是通过析构函数 3. 析构函数会递减引用计数，若为0就会销毁对象。并释放它占用的内存 4. Shared\_ptr还会自动释放相关联的内存 5. 嗯，就是5.1说的 6. 在最后一个shared\_ptr销毁前内存都不会释放，若有不在需要的shared\_ptr，程序正确执行，但浪费内存。应该用erase删除那些不再需要的shared\_ptr元素 7. 如：shared\_ptr存放在一个容器中，随后重排了容器，不再需要某些元素，只需要部分，所以需要erase删除不再需要的元素 8. 使用了动态生存期的资源的类   使用动态内存的原因：   1. 程序不知道自己需要使用多少对象 2. 程序不知道所需对象的准确类型 3. 程序需要在多个对象间共享数据 4. 原因1：容器类 5. 原因2：继承时 基类指针需指向子类指针才会发生动态绑定 6. 原因3：就是shared\_ptr，可以自定义一个类有动态内存的对象来记录多少个共享对象，实现类似shared\_ptr功能 7. 我们所用的类中，分配的资源都与对象对象生存期一致。如：vector赋值给一个vector，其实拷贝vector中的元素，销毁一个vector不影响另一个vector。   我们可以实现类，分配的资源具有与源对象相独立的生存期。如：一个类赋值给一个类对象后，左侧对象引用右侧对象的值，当右侧对象销毁，这个值在内存中不该销毁。  Note：使用动态内存是允许多个对象共享相同的状态。   1. 定义StrBlob类 2. 实现这个类上面所说的不是拷贝，是引用，不会删除底层数据 3. 使用shared\_ptr管理动态分配的vector…等一些方法   10. StrBlob构造函数  1) 用make\_shared初始化shared\_ptr，参数可以是一个列表initializer\_list  11. 元素访问成员函数  12. StrBlob的拷贝、赋值和销毁  1) 使用默认版本。Shared\_ptr会自动处理什么时候删除底层数据。构造是拷贝引用不是拷贝底层数据 | |



