|  |  |
| --- | --- |
| 1. 直接管理内存 2. 使用new动态分配和初始化对象 3. 动态分配的const对象 4. 内存耗尽 5. 释放动态内存 6. 指针值和delete 7. 动态对象的生存期直到被释放时为止 8. Delete之后重置指针值 9. 这只是提供了有限的保护 | New  Delete表达式  Int \*p1 = new int;  Int \*p2 = new int(1034);  Vector<int> \*pv = new vector<int>{1,2,3};  Int \*p3 = new int();  Auto p4 = new auto(obj);  Auto a5 = new auto{a,b,c};//错误  Const int \*pci = new cons tint(1024);  Int \*p1 = new (nothrow)int;  定位new  delete p1;  Foo\* factory(T arg)  {  Return new Foo(arg);  }  Void use\_factory(T arg)  {  Foo \*p = factory(arg);  Delete;  }  空悬指针  Auto q = p;  Delete p;  P = nullptr; |
| 1. 直接管理内存 2. New分配内存，delete释放new分配的内存 3. 不能依赖类对象拷贝 赋值和销毁操作的任何默认定义   Note：最好用智能指针分配内存   1. 使用new动态分配和初始化对象 2. 自由分配的内存是无名的，因此new无法为其分配的对象命名，而是返回一个指向该对象的指针 3. 默认下动态分配的对象是默认初始化的，内置类型或组合类型的对象的值将是未定义的，而类类型对象将用默认构造函数进行初始化 4. 可以使用直接初始化方式、列表初始化、传统的构造方式   int \*p3 = new int(10);// 直接  string \*pstr2 = new string(10, '9');//构造  vector<int> \*pv1 = new vector<int>{ 1,2,3 };//列表   1. 值初始化，类型名后加一个括号。   string \*pstr3 = new string();// 与默认构造函数的默认初始化一样  int \*p4 = new int();// 内置类型 有区别，默认初始化为未定义，值初始化良好定义的值。在勒种依赖合成的默认构造函数内置类型成员，若他们未被在类内初始化，他们的值也是未定义的。   1. Note：处于与变量初始化相同的原因，对动态分配的对象进行初始化通常是个好主意 2. auto推断类型，只有单一初始化器时才可以。若obj是int，p1是int\* 3. 动态分配的const对象 4. 必须初始化。一个定义了默认构造函数的类类型，其const动态对象可以隐式初始化，而其他类型的对象必须显式初始化。分配的对象是const的，返回的指针是一个指向const的指针   const int \*p7 = new const int;// 未定义 也可以？14？  const string \*p8 = new const string;// 隐式初始化   1. 内存耗尽 2. 堆内存耗尽可能发生。耗光了再new会抛bad\_alloc。可以改变new的方式阻止异常 3. 定位new，若异常不能new出来则返回一个空指针。Bad\_alloc与nothrow在头文件new中   int \*p9 = new (nothrow) int; // 定位new 允许我们向new传递额外的参数   1. 释放动态内存 2. 释放动态内存 将内存归还给系统，两个动作，销毁给定的指针指向的对象，释放对应的内存   delete p9;// 指向的可以是空指针 或者 动态分配的对象   1. 指针值和delete 2. 删除的对象空指针 或者 动态分配的内存 3. 若多次删除同一个指针值或不是new分配的内存，未定义 4. 编译器不能分布一个指针指向静态的还是动态的对象。同样也不能分辨一个指针所指的内存是否已经被释放了。Delete表达式编译器会通过，尽管他们是错误的 5. 编译通过，但是却是错误的 6. 动态对象的生存期直到被释放时为止 7. 内置指针管理的动态对象,直到被显式释放之前都是存在的 8. 函数返回一个指针，那么调用者要记得释放，但是调用者经常忘记释放 9. 即使所在作用域的函数结束了，指针会销毁 但是指针所值得内存不会销毁不会自动释放   Note:动态内存的管理非常容易出错  1) 内存泄漏，当内存很多未delete。内存泄漏在程序运行很长一段时间后才能检测这种错误，查找内存泄漏错误是非常困难的。  2) 使用已经释放掉的对象。有时在释放内存后将指针置位空，有时可以检测出这种错误  3) 同一个内存释放两次。   1. Delete之后重置指针值 2. delete一个指针后，指针值无效。但很多机器上指针仍然保存值已经释放了的动态内存的地址。空悬指针：指向一块曾经保存数据对象但现在已经无效的内存的指针 3. 具有未初始化指针所有缺点 4. 解决： 5. 在即将离开作用域时释放掉它所关联的内存 6. 若想保留指针，在delte之后将nullptr赋予指针。 7. 这只是提供了有限的保护 8. 上面的解决方法2。单对这个指针有效，若多个指针指向相同的内存，那么对其他指针是没有作用的。 9. 删除其一指针释放内存，再赋为nullptr使其安全，但是其它指针却还是未定义的   int \*p15 = new int(12);  int \*p16 = p15;// 指向相同的内存  delete p15;// p  p15 = nullptr;// 指出p不再绑定到任何对象,但p16并不为nullptr为空悬指针，也需要指向nullptr | |