智能指针

自动释放，异常安全，value语义转化为reference语义（Java）。value语义依赖于赋值和赋值如返回一个值，对oop不便，强行解决使用指针会产生释放问题，使用reference语义解决即智能指针。

unique\_ptr::release()：置空不释放，返回基本指针，所以不使用返回值的release()调用铁定是错。.reset()置空释放。

切勿用智能指针引用栈内存，引起错误释放。

禁止普通指针和智能指针之间的隐式类型转换。

auto\_ptr和unique\_ptr采用单一所有权策略解决释放问题，unique\_ptr严格更好。永远不要用auto\_ptr！auto\_ptr允许智能指针间赋值，实质为转移所有权。交出者空置，遗留危险。unique\_ptr改进：当一个 unique\_ptr 赋值给另一个时，如果源是个临时右值（如返回值），编译器允许（unique\_ptr<std::string> foo = makefoo()）；如果源将悬空存在一段时间，编译器禁止；确实需要悬空存在行为的，使用move显式转移（允许情形本质就是自动调用move）。move后右值行为未定义。

可用unique\_ptr构造shared\_ptr，赋值条件相同，所有权转移。

unique\_ptr记住一件事：永远只有一个，转移必释放。所以若做返回值，返回std::move()；要copy到另一个unique\_ptr，= std::move()。

shared\_ptr对赋值的要求少所以比较好用。对unique\_ptr，stl中涉及赋值的算法不可用（如sort）。参数unique\_ptr的引用没问题，参数unique\_ptr则实参必须为临时右值或std::move()，即显式移交所有权。

weak\_ptr依附于shared\_ptr，从shared\_ptr或weak\_ptr构造。对象被shared\_ptr销毁时置expired。使用前看expired。不可用weak\_ptr调用方法。需shared\_ptr = weak\_ptr::lock()，再用它调用方法。理由：增加强引用计数，确保使用时资源不被销毁。

unique\_ptr天然可用于数组。shared\_ptr默认使用delete释放所以不可用于数组，除非手动指定释放方法为delete[]（shared\_ptr<int> ptr(new int[5], [](int\* p){delete[] p;});）。

不要用一个基本指针构造多个智能指针！他们互相不了解，均认为自己拥有并负责释放。

多线程环境中使用shared\_ptr的代价非常大。这是因为需要避免关于引用计数的数据竞争。