unique\_ptr：

1.使用方法：std::unique\_ptr<T> ptr(new T()); // 创建一个指向动态分配的T类型对象的独占指针

2. 特点：std::unique\_ptr是独占所有权的智能指针，一个对象只能被一个std::unique\_ptr所拥有；当std::unique\_ptr被销毁或重置时，它所指向的对象会被自动删除；不可以进行所有权的安全转移，拷贝std::unique\_ptr会导致原指针失效。

3.使用场景：适合用于需要独占对象的场景，如临时对象或需要确保只有一个指针指向对象的场景。

shared\_ptr：

1.使用方法：std::shared\_ptr<T> ptr = std::make\_shared<T>(); // 创建一个指向动态分配的T类型对象的智能指针

2.特点：多个std::shared\_ptr可以共享同一个对象的所有权，当最后一个指向对象的std::shared\_ptr被销毁或重置时，对象会被自动删除；使用引用计数机制，通过一个内部计数器跟踪指向对象的指针数量；可以进行安全的转移所有权，通过std::shared\_ptr的拷贝构造函数或std::make\_shared函数；

3. 使用场景：适合用于需要多个指针指向同一个对象的场景，如多个成员变量或函数返回多个指针指向同一对象。

weak\_ptr：

1.使用方法：std::weak\_ptr<T> ptr = std::make\_weak<T>(); // 创建一个指向动态分配的T类型对象的弱指针

2. 特点： std::weak\_ptr不会拥有对象的所有权，它只是观察一个已经被std::shared\_ptr拥有的对象；std::weak\_ptr不会增加引用计数，因此不会阻止对象的删除；当对象被删除后，std::weak\_ptr会变成空的，尝试访问空指针会导致未定义行为。

3.使用场景：适合用于需要观察共享对象的生命周期，但不需要拥有对象所有权的场景，如观察者模式或回调函数。