一、多态

1.多态的概念

多态:同一个事物，在不同场景下表现出的不同的状态

举例子来说明:见人说人话，见鬼说鬼话　学生买票的例子　自己想一些其他例子

2.多态的分类

静态多态(早绑定，静态联编): 在编译期间，根据所传递的实参类型或者实例化的类型,来确定到底应该调用那个函数即:在编译期间确定了函数的行为---函数重载、模板

动态多态(晚绑定,动态联编):在程序运行时，确定具体应该调用那个函数

3.动态多态的实现条件-- -在继承的体系中

>>虚函数&重写:基类中必须包含有虚函数(被virtual修饰的成员函数)，派生类必须要对基类的虚函数进行重写

>>关于虚函数调用:必须通过基类的指针或引用调用虚函数

体现:在程序运行时，基类的指针或引用指向那个子类的对象，就会调用那个子类的虚函数

4.重写

>> 1.基类中的函数一定是虚函数

>> 2.派生类虚函数必须与基类虚函数的原型一致：返回值类型 函数名字(参数列表)

例外:协变--基类虚函数返回值基类的指针或引用

派生类虚函数返回派生类的指针或引用基类虚函数和派生类虚函数的返回值类型可以不同

析构函数:如果将基类中析构函数设置成虚函数,派生类的析构函数提供，两个析构函数就可以构

成重写；两个析构函数名字不同

>>3.基类虚函数可以和派生类虚函数的访问权限不一-样

1>C+ +中构成重写的条件非常严格:虚函数 并且 原型一致

基类虚函数: virtual void TestFunc()

派生类虚函数: virtual void TetsFunc();//细节性的不同不容易发现，而导致重写失败

2>为了让编译器在编译期间帮助用户检测是否重写成功，C+ +11提供非常有用的关键字override:专门]让编译帮助用户检测派生类是否成功重写了基类的虚函数

如果重写成功:编译通过

如果重写失败:编译失败

final:如果用户不想要子类重写基类的虚函数,可以使用final修饰该关键字

3>函数重载、同名隐藏(重定义)、重写(覆盖)之间的区别?

函数重载概念:相同作用域、函数名字相同、参数列表不同(类型、个数、类型次序)

同名隐藏　　重写

相同点: 一个函数在基类中，一个函数在子类中

不同点:基类中函数是否为虚函数?

没有要求　　一定要是虚函数

函数原型是否相同?

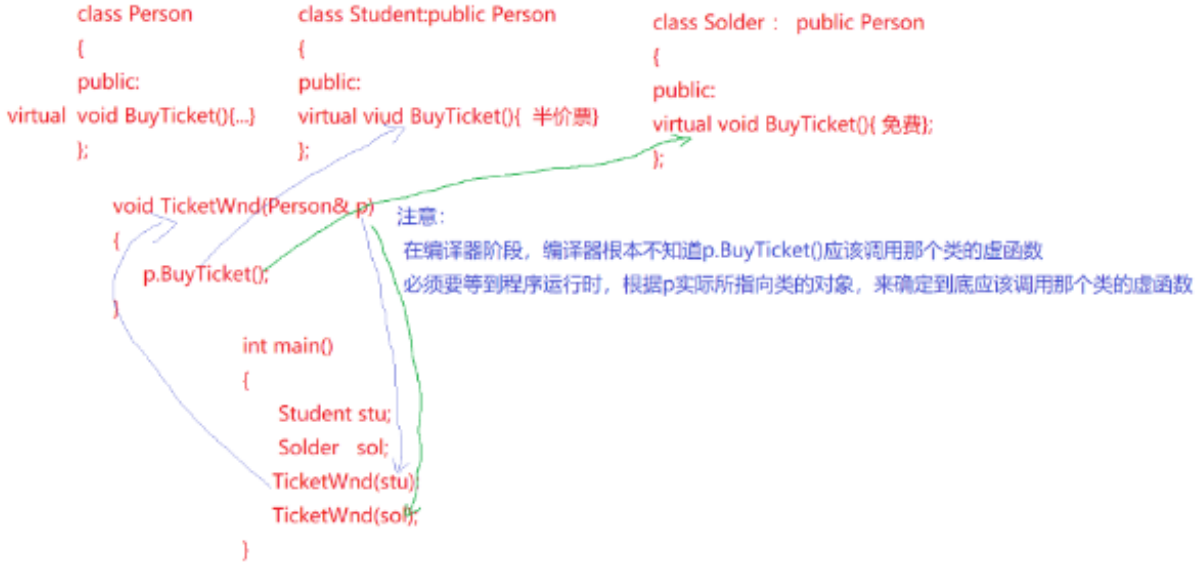
只要名字相同即可，其他没有要求　　原型必须要相同

重写的限制条件比同名隐藏更加的严格:

如果满则同名---但是没有满足重写的条件---定是同名隐藏

注意:同名隐藏和重写是两个不同的概念,不能说同名隐藏是一种特殊的重写

5、举例子说明

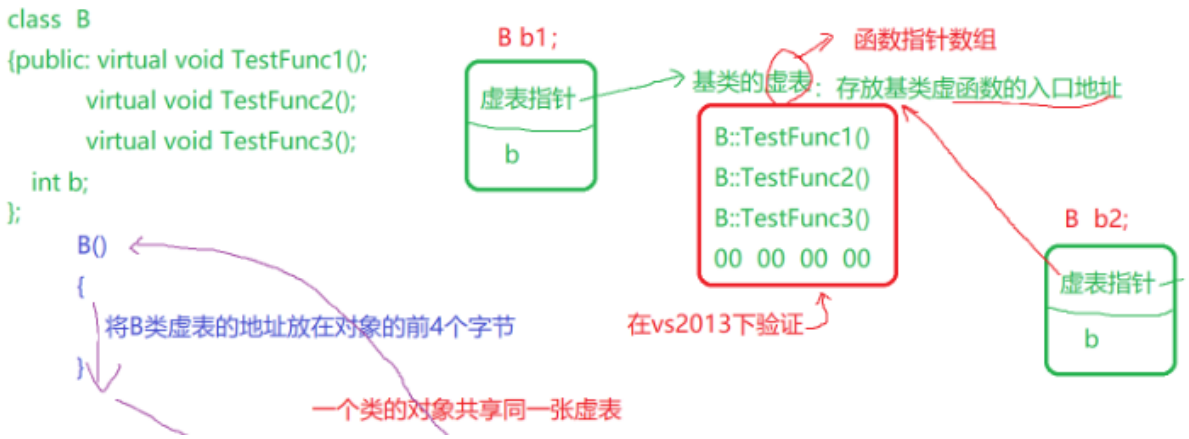


6、多态的实现原理

编译器在编译时会将类中的虚函数按照一定的规则存储在虚表中，在创建对象时，只需要将虚表的地址存储在对象的前四个字节

>>虚表构建过程

1、基类：编译器按照各个虚函数在类中声明的先后次序依次将虚函数保存在虚表中

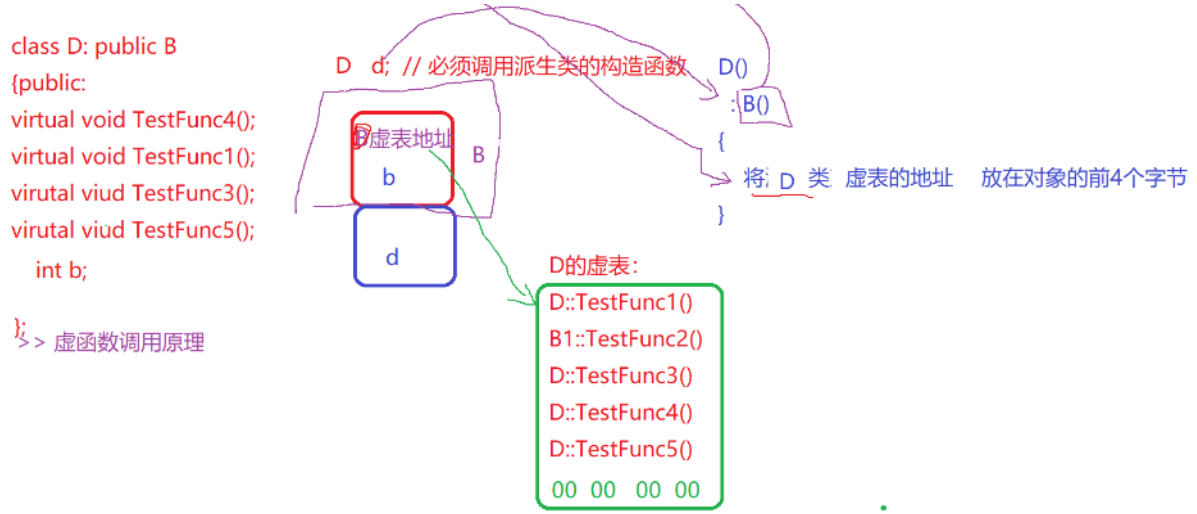


2.子类虚表的构建过程

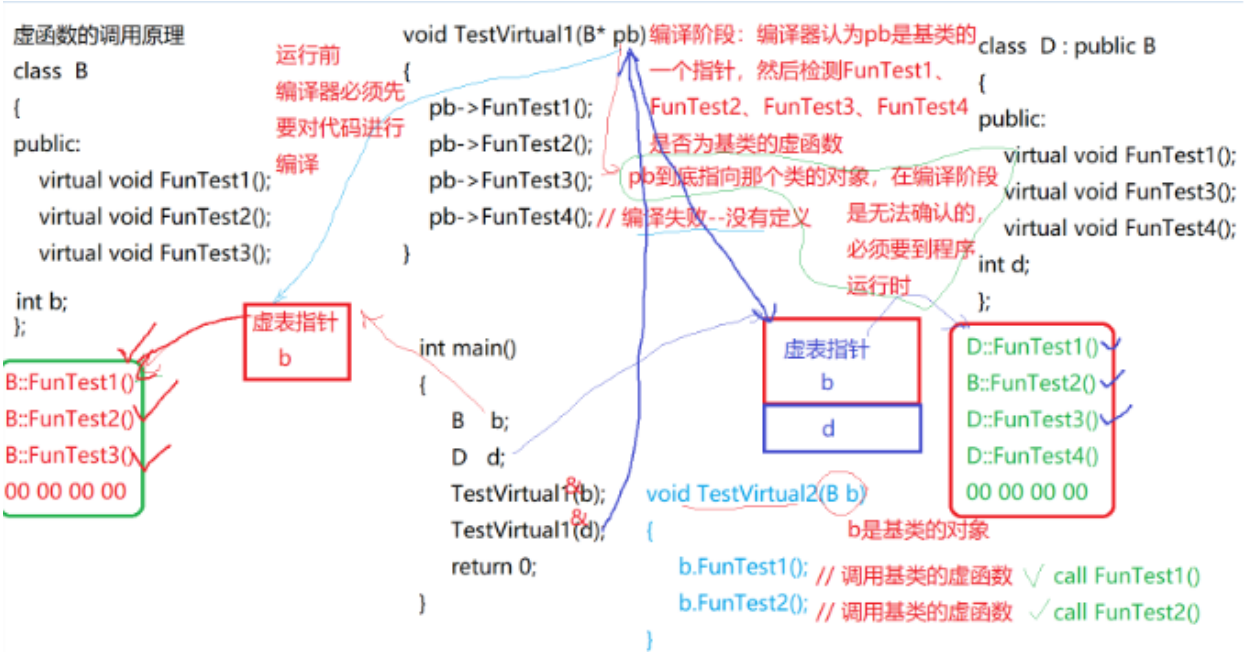
a.将基类虚表中内容拷贝一份到子类虚表中

b.如果子类重写了基类中那个虚函数，编译器会用子类自己虚函数的地址覆盖相同偏移量位置的基类虚函数地址

c.如果派生类新增加自己的虚函数，编译器会将派生类新增加的虚函数按照其在派生类中的声明次序依次放在虚表的最后



>>虚函数调用原理



虚函数调用---

1.如果虚函数是通过基类对象直接调用----多态的条件没有完全完满---直接调用基类的虚函数

2.如果虚函数是通过基类的指针或引用调用---多态的条件已经满足

虚函数调用步骤:

a.从对象前4个字节中获取虚表的地址(找虚表)

b. 从虚表中获取当前虚函数的入口地址

C. 虚函数传参

d. 调用虚函数

如果通过基类的指针或者引用调用虚函数，不论子类是否对基类的虚函数进行重写,编译器对虚函数的调用都会采用虚表的方式进行调用

```
int mian()
{
  D d;
  B* pb=(B*)&d;
//此处强转具有一定的迷惑性， 实际pb指向的还是子类的对象，因此最终从子类的虚  表中取到FunTest1虚函数的入口地址，进行调用
  pb->FunTest1();
}
```

7、多态缺陷:

1.类会背负一份比较大的虚表，浪费空间

2.虚函数比常规函数调用速度慢-- -因为虚函数需要一步一 步的找虚函数的地址--- 会降低程序运行的效率

8、抽象类:

1>.抽象类的概念：将包含有纯虚函数的类称为抽象类，纯虚函数---在基类中定义虚函数时，如果该虚函数的行为无法确定，只需要给出该虚函数接口=0

2>.抽象类的特性

抽象类不能实例化对象----抽象类可以将其看成是一个不完整的类

子类必须要对基类中纯虚函数进行重写，才可以创建对象，否则：子类也是一个抽象类

6. 多态常见的面试问题

1. 什么是多态？答：参考本节课件内容
2. 什么是重载、重写(覆盖)、重定义(隐藏)？答：参考本节课件内容
3. 多态的实现原理？答：参考本节课件内容
4. inline函数可以是虚函数吗？答：不能，因为inline函数没有地址，无法把地址放到虚函数表中。
5. 静态成员可以是虚函数吗？答：不能，因为静态成员函数没有this指针，使用类型::成员函数的调用方式无法访问虚函数表，所以静态成员函数无法放进虚函数表。
6. 构造函数可以是虚函数吗？答：不能，因为对象中的虚函数表指针是在构造函数初始化列表阶段才初始化的。
7. 析构函数可以是虚函数吗？什么场景下析构函数是虚函数？答：可以，并且最好把基类的析构函数定义成虚函数。参考本节课件内容
8. 对象访问普通函数快还是虚函数更快？答：首先如果是普通对象，是一样快的。如果是指针对象或者是引用对象，则调用的普通函数快，因为构成多态，运行时调用虚函数需要到虚函数表中去查找。
9. 虚函数表是在什么阶段生成的，存在哪的？答：虚函数是在编译阶段就生成的，一般情况下存在代码段(常量区)的。
10. C++菱形继承的问题？虚继承的原理？答：参考继承课件。注意这里不要把虚函数表和虚基表搞混了。
11. 什么是抽象类？抽象类的作用？答：参考（3.抽象类）。抽象类强制重写了虚函数，另外抽象类体现出了接口继承关系。

12. 继承(is-a) 和 组合(聚合) has-a

```
class B
{
public:
virtual  ~B(){....}
};
```

```
class D : public B
{
public:
  D(){pd = new int[10];}
  ~D(){ delete[] pd; }
int* pd;
};
```

场景：子类中包含有动态资源的管理

基类的析构函数最好(一定)设置成虚函数

否则：可能会存在资源泄漏

```
int main()
{
   B* pb = new D;
   delete pb;
}
```

1. 调用析构函数释放对象中的资源

pd是基类的指针(指向的是子类对象)，本应该调用子类的析构函数资源是合理的，但是因为基类中的析构函数不是虚函数，那么子类的析构函数就不会重写基类中的虚函数，多态的条件没有完全满足，最终调用基类的析构函数，pb指向派生类对象中的资源就无法释放而引起资源泄漏

2. 调用operator delete(p)释放空间

二 、智能指针: *****

1.为什么需要有智能指针?

指针:灵活　缺陷:用户动态申请资源- - -通过指针接收　手动释放--- >容易被遗忘掉||代码丑陋

能否让动态自动去进行释放

2.什么是RAII?

资源获取即初始化---在C++中，创建对象或销毁对象时,编译器会自动调用构造函数完成对象初始化,会自动调用析构函数完成对象中资源的清理工作

可以巧妙借助构造和析构

3.如果让你设计智能指针，都需要实现哪些方法?

智能指针主要职责:帮助用户管理资源，并在合适的时机释放资源

智能指针:

1>.包含一个指针--->T*(可以通过模板方式)

2>. RAII:构造中将用户的资源进行接收，在析构中将资源释放掉

3>.让智能指针的对象具有指针类似的行为---operator*() / operator->()

4>.解决浅拷贝问题:用户必须显式实现拷贝构造函数以及赋值运算符重载

4.各个智能指针的实现原理以及区别

C++98: auto_ptr

<一>auto_ ptr: RAII + operator*0/operator->() +解决浅拷贝的方式

解决浅拷贝方式:

1.资源的转移auto_ptr<int> ap1(new int); auto_ptr<int> ap2(ap1); ap1将其管理的资源直接转移ap2,然后ap1与资源完全断---->效果:可能让一个资源只释放一次， 那个对象最终拥有资源，那个对象进行释放缺陷: ap1和ap2不能同时拥有资源

2.资源管理权的转移(只转移资源释放的权利，而需要对资源断开关联)

在类中增加bool类型的成员变量:

owner-->true: 表示当前对象在其声明周期结束时，必须要释放资源

false:表示当前对象不能释放资源

auto_ptr<int> ap1(new int); auto_ ptr<int> ap2(ap1);

ap1和ap2共享同一份资源，ap1没有与资源断开联系，解决方式1中的缺陷:

新的缺陷:可能会造成野指针

```
void TestPtr()
{
    auto_ ptr<int> ap1(new int); // owner  true
    if(true)
        auto ptr<int> ap2(ap1); //  ap1:owner false ap2:owner true
//ap1没有与资源断开联系出了if的块, ap2对象将要被销毁，其在销毁时已经将空间释放,ap1根本不知道资源已经被释放掉ap1实际已经是一 个野指针
    *ap1 = 10; .//此时ap1已经是野指针--继续访问其关联的空间，会引起代码崩溃
}
```

C++11---又将实现原理退回到方式1

标准委员会建议:什么情况下都不要使用auto ptr

具体代码实现---参考课堂代码

C++11---提供几个可以正常使用的智能指针

<二>unique_ ptr: 一个资源只能被一个unique_ ptr类型的对象进行管理，禁止多个对象之间共享资源

实现原理: RAII + operator*(/operator->() +防拷贝

防拷贝:

C++98:只需要将拷贝构造函数以及赋值运算符重载只声明不定义&必须将其访问限定符设置为private

C++11:只需要在拷贝构造函数和赋值运算符重载= delete (说明: =delete编译器不要生成默认的拷贝构造以及赋值运算符重载)

unique_ ptr可以正常使用，唯一不足之处: 不能共享资源

<三>shared_ ptr:因为unique_ptr不能满足所有的场景，因此提供shared_ptr

实现原理: RAII + operator*()/operator->() +解决浅拷贝方式

解决浅拷贝方式:采用引用计数的方式解----引用计数:资源被共享的对象的个数

引用计数的方式:

1. 普通类型的整形成员变----不可以:计数需要被多个对象共享，每个对象中都包含一个计数， 如果一个对象将计数更改，并不会影响其他的对象

2.静态整形成员变----不可以:静态成员变量是所有类对象共享，引用计数并不是所有对象共享，引用计数记录的是使用资源的对象个数，引用计数应该是和资源挂钩，有多少分资源就应该有多少个计数,共享同一份资源的对象之间共享同一份计数

3.整形的指针

具体代码---可以参考课堂代码

优点:大部分场景都可以处理

缺陷:可能会存在循环引用问题

1.什么是循环引用

-1、node1和node2两个智能指针对象指向两个节点，引用计数变成1，我们不需要手动delete。

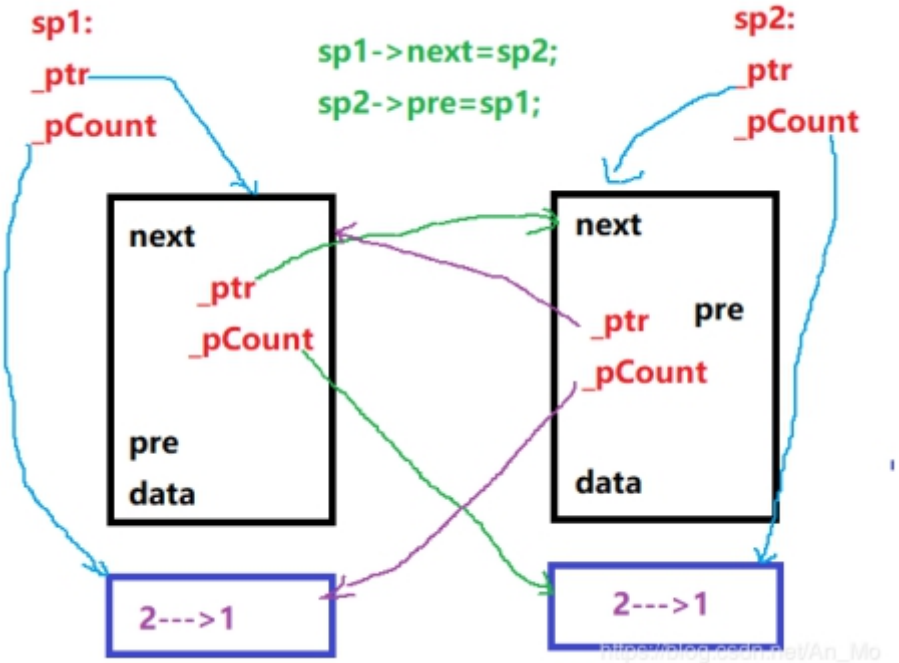
- 2、node1的_next指向node2，node2的_prev指向node1，引用计数变成2。

- 3、node1和node2析构，引用计数减到1，但是_next还指向下一个节点。但是_prev还指向上一个节点。

- 4、也就是说_next析构了，node2就释放了。

- 5、也就是说_prev析构了，node1就释放了。

- 6、但是_next属于node的成员，node1释放了，_next才会析构，而node1由_prev管理，_prev属于node2成员，所以这就叫循环引用，谁也不会释放。



2.循环引用会造成什么后果

资源没有释放--到引起资源泄漏

3.循环引用如何解决

//

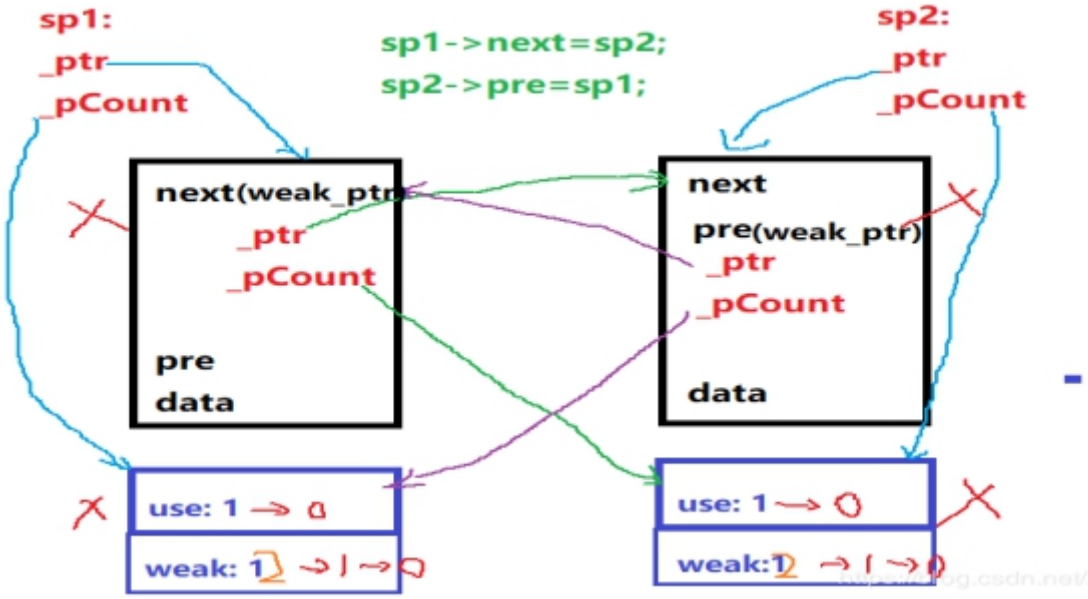
解决方式：在引用计数的场景下，把节点中的_pre和_next改成weak_ptr就可以了

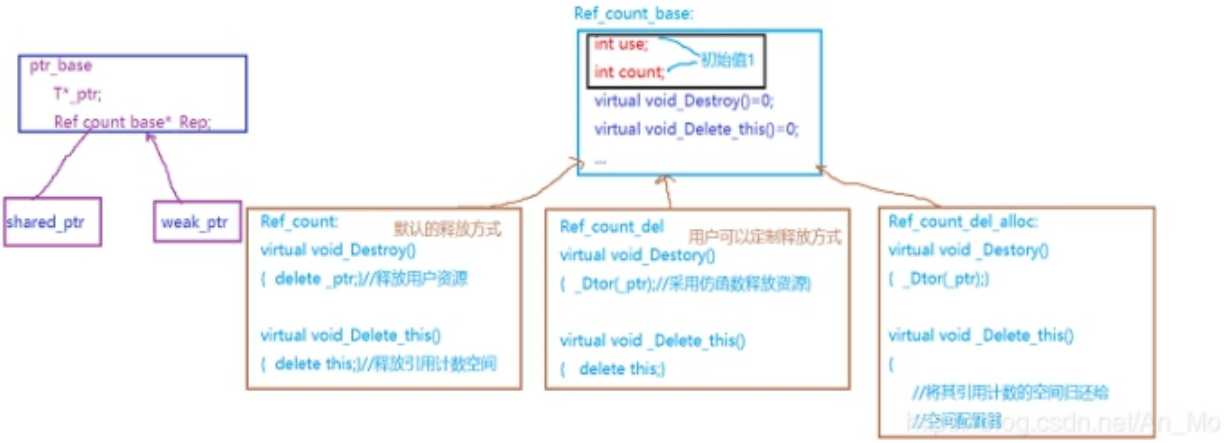
weak_ptr:

1）实现原理：RAII+具有指针类似操作+引用计数

2）作用：配合shared_ptr，解决其循环引用问题

3）注意：weak_ptr对象不能独立管理资源



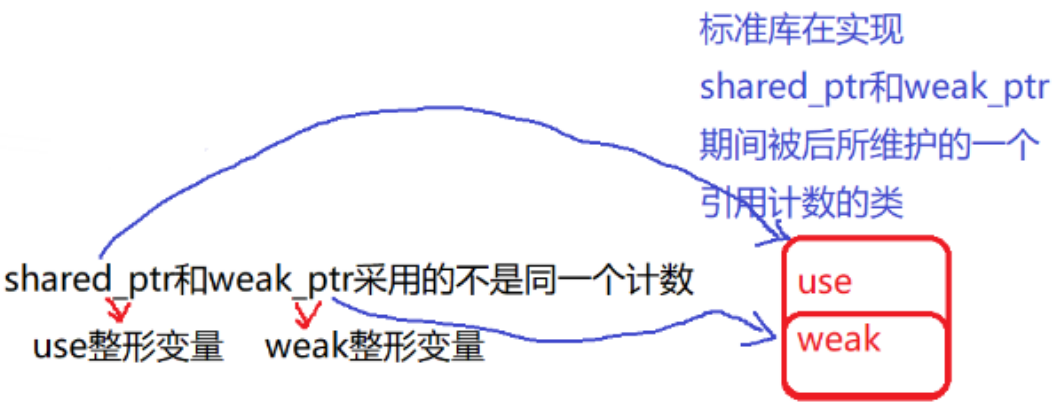


//

weak_ ptr:作用-配合shared _ptr使用，帮助shared_ptr解决循环引用的问题

weak_ ptr不能独立的管理资源

实现原理:与shared_ptr的实现原理一致，都采用引用计数shared_ptr和weak_ptr采用的不是同一个计数



定制删除器:
为什么要定制删除器:资源的种类比较多，不同类型的资源应该选择合适的方式进行释放，因此智能指针的析构函数中不能将资源的释放方式写死，不同类型的资源就会采用同一种方式进行释放而引起代码崩溃
定制方式:在实现智能指针时，给用户预留选择释放方式的接口
>> 1.多给一一个模板类型的参数---上课讲
>> 2.可以通过参数
具体实现: 1. 函数指针2. 仿函数3. lambda表达式

5.智能指针和STL中容器的区别
C++11提供的智能指针只能管理单个独享的空间，不能管理一段连续的空间
为什么: STL中已经存在vector
智能指针:管理的资源是外部用户申请的
STL容器:资源是容器自己维护
6.面试官可能会让学生模拟实现一个智能指针
建议:
1.模拟实现unique. ptr,因为:实现简单
2.最好可以定制删除器---参考: shared. ptr定制删除器方式