2016年10月10日 18:23

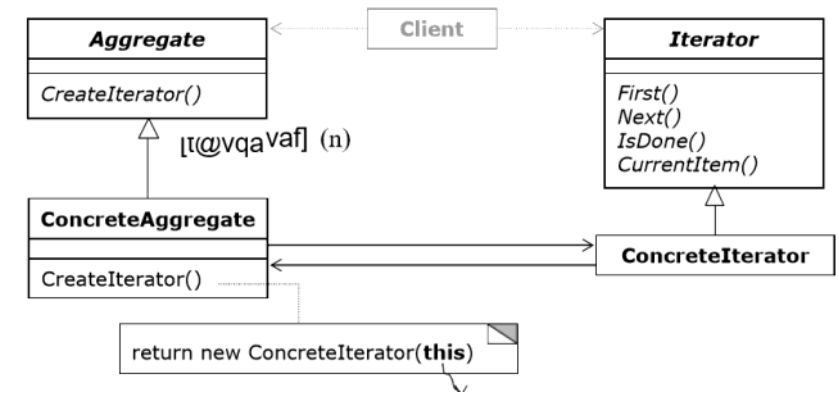
Reference:
C++中reference就是一种形式很好的pointer。因为它并不需要指针所需要的*和->。
Java中所有的变量都是refernece。
{reference是一种安全的指针，即没有指针运算的指针。}

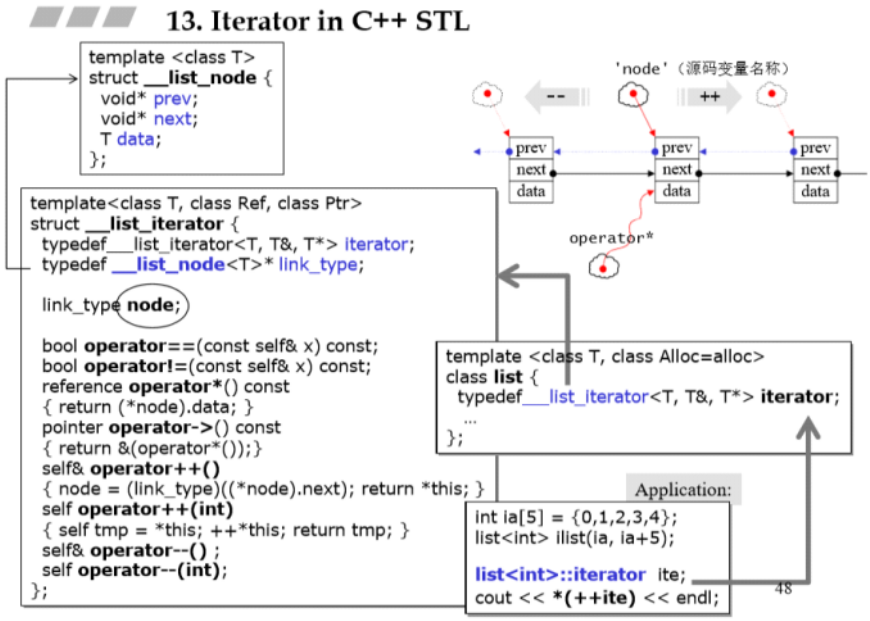
Class之间的关系：
1.Composition(复合): 拥有。A<>------>B ， A拥有B，二者生命周期一致。
2.Delegation(委托): 指针指向。A和B的生命周期不一致。 （可以用指针指向一个继承体系的父类，这样这个指针就有弹性了。比如 HumanObj 指向 AnimalObj, 而AnimalObj可以是牛羊的父类。 ）
3.inherit(继承):实际上是继承了调用权

守则：

设计模式：
1.Template Method of GOF(模板方法):
定义算法的骨干，延缓其中的某些步骤，使他们在subclass中获得重新的定义。
2.Observer(观察者):
在 objects 之间定义 ″一对多″ 相关性，使得当 object 改变状态时，它所 依存的所有 objects 都会获得通知并自动更新。
"一"是被观察者， "多"是观察者。 观察者是被动的通知，而不是主动观察。
被观察者提供注册和注销两个功能，而观察者要提供被观察者可以调用的函数。
可以用继承来实现观察者模式， 子类是观察者， 父类是被观察者。
3.Iterator(迭代器):
提供一种连续（相继）巡访「聚合物内各元素」的通用接口，并且不 必曝露聚合物的底层表述（内部细节）。

Java 的实现：



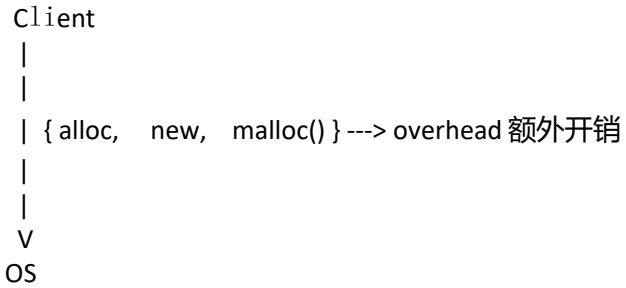


C++中Iterator不同的STL有不同的实现，该版本直接把Iterator作为结构体放入了目标的容器中。

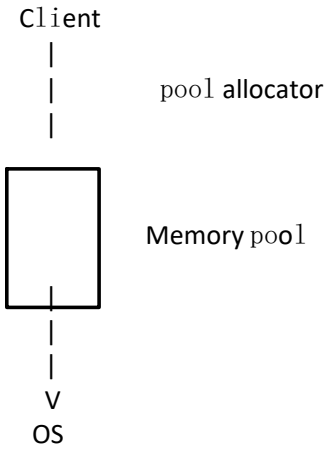
4.重载operator new 和 operator delete:

memory pool : 分配一块内存再细切，为客户（容器）服务。内存分配方式：

1.



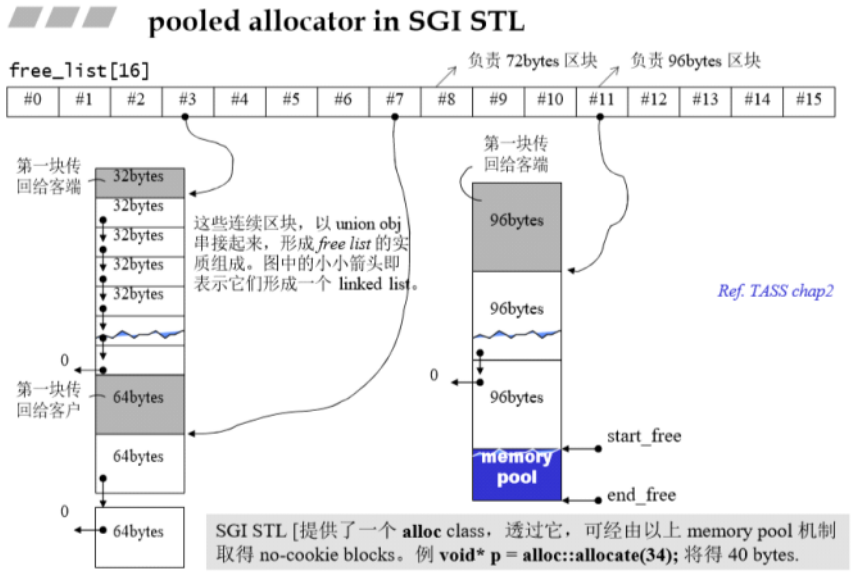
2.



容器用内存分配器：

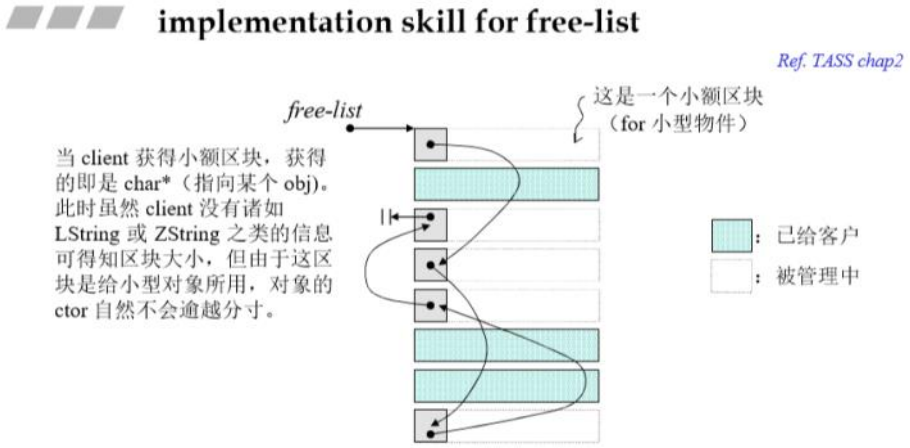
下例中，每个指针负责管理（8 * #n ）大小的内存块。所以容器向分配器要求内存实际上是要求内存服务指针。每次容器请求内存，allocator会多分配出很多内存（例如要20个），第一个返回给客户，多余的以链表进行组织备用。如果超过最大的服务指针最大限制，就直接调用malloc()。

改进： 可以要20个的两倍 -- 40个， 但是只切20个，剩下的留给其他内存大小的申请请求。（如图中32byte和64byte）。如果不能除尽的话会进行碎片的处理。



释放的细节：

如果不是逆序释放，有可能会变成下图所示， 但是由于是链表所以没关系。



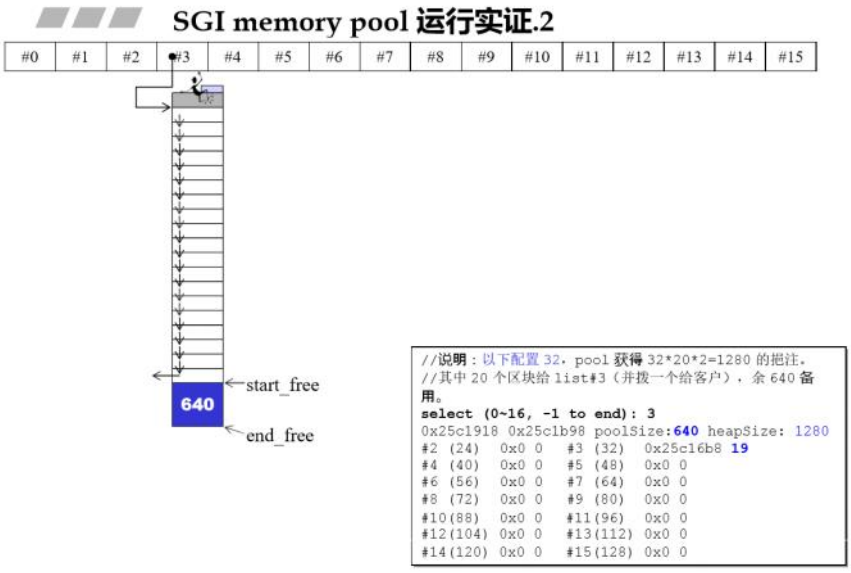
内存池： 1. 减少了System Call

2. 减少Overhead { 实际上， malloc() 的内存会带一个头--cookie_1 + debug header(只有debug的时候才有) + memory + cookie_2， cookie_1记录了长度 }, 在上述的细切中，是不含有cookie的，因此减小了额外开销

```
/*
 注：用法  GNUC
      void * p4 = alloc::allocate(512);
      alloc::deallocate(p4,512);
 不鼓励这么做，因为分配器是给容器设计的。
 传大小：分配器是给容器使用的，容器的内存大小是固定的，所以记录大小很方便。
 而malloc是给所有人用的，所以都有cookie，需要用cookie记录大小。
*/
```

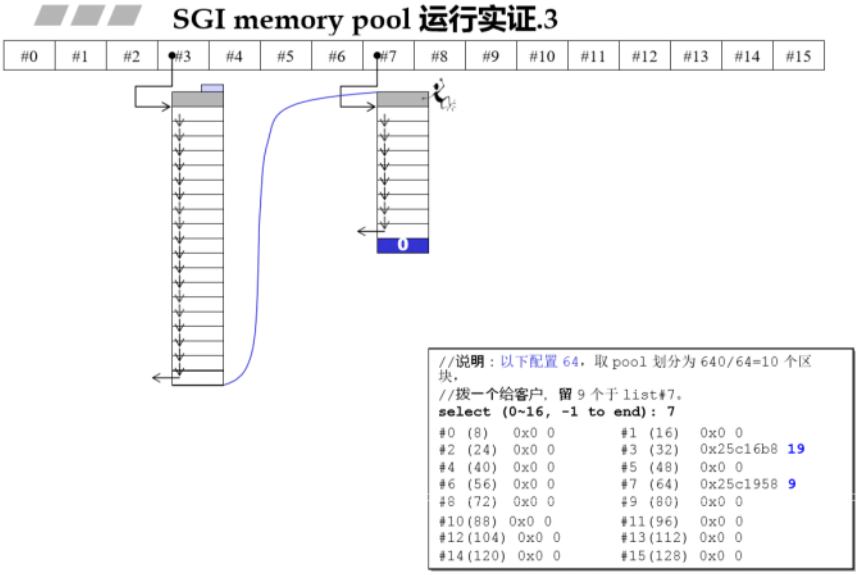
调用过程：

2. List<T> v; v.push_back(T()); sizeof(T) == 24 + 2 * sizeof(ptr) == 32



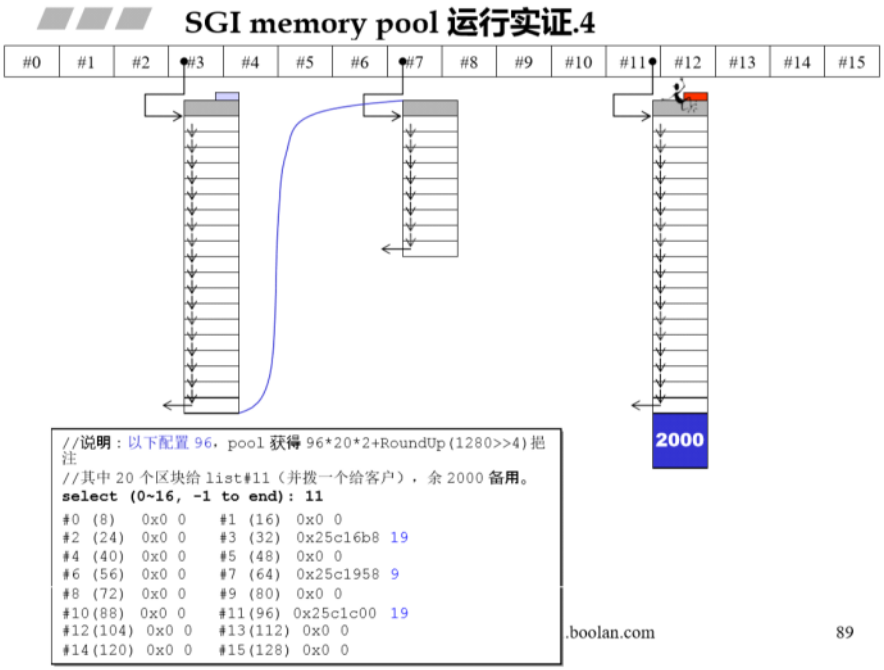
开头的小蓝色部分是Cookie

3. List<T2> v2; v2.push_back(T2()); sizeof(T2) == 56 + 2 * sizeof(ptr) == 64



#7没有Cookie是因为是直接从战备池拿出来的

4. 再分配 96 的内存

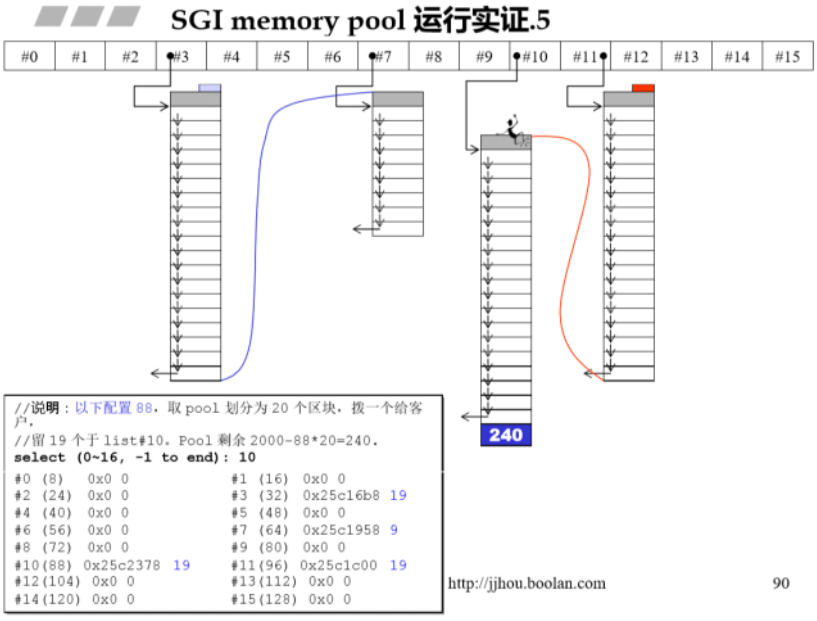


RoundUp---追加量 ：根据在此的操作酌情在增加分配的内存量。

1280是这个动作之前总共分配的内存大小

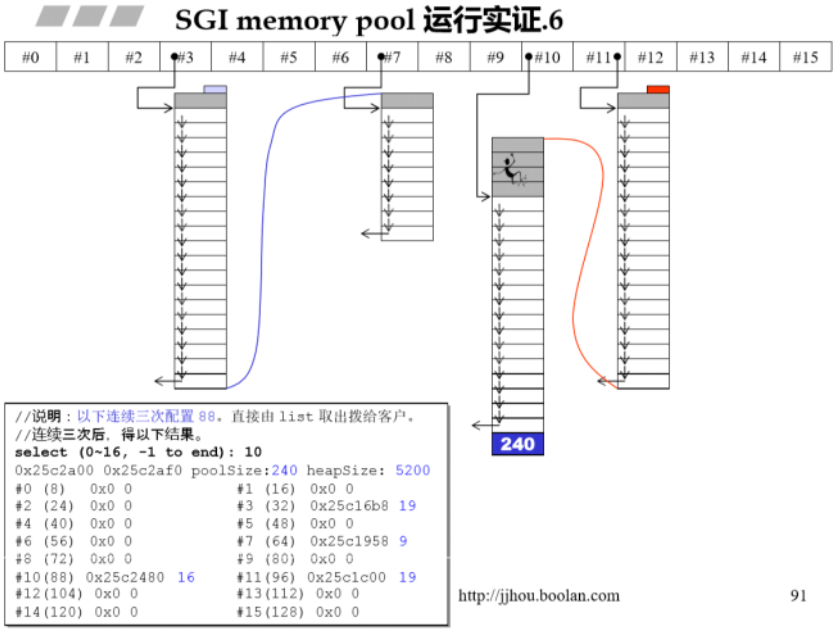
此次分配带有cookie

5. 又申请了88个字节

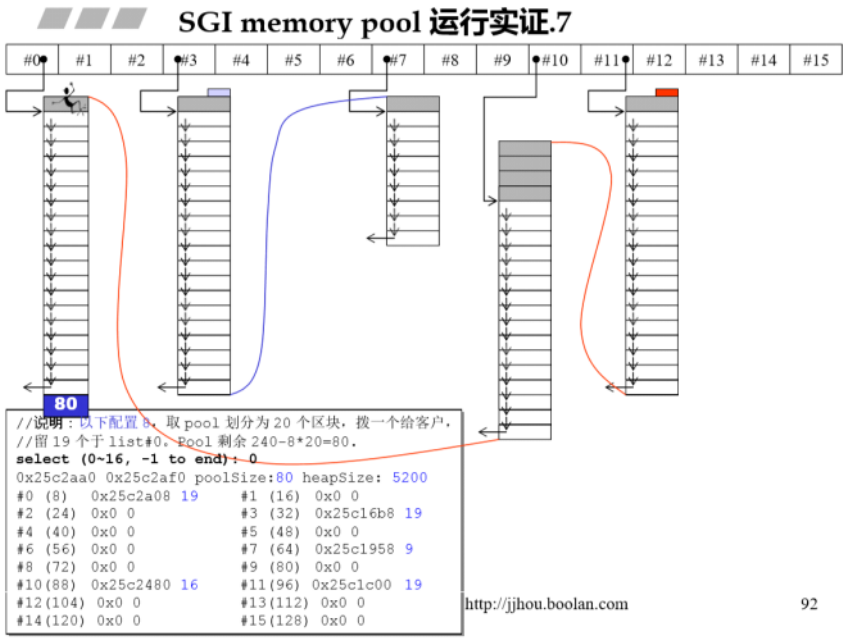


这次没有cookie是因为是由之前申请96byte的内存池分过来的。

6. 连续分配三个88byte

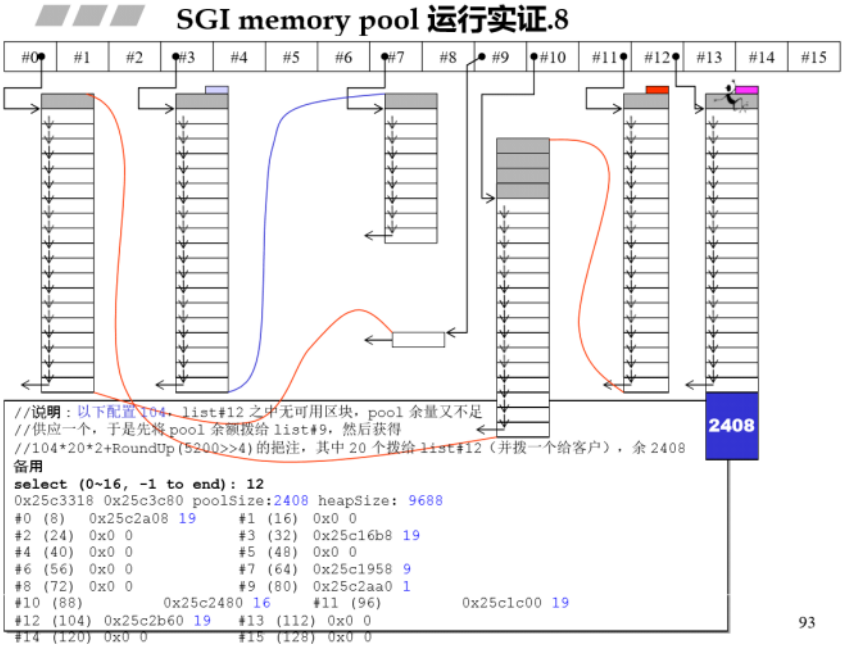


7.申请8byte



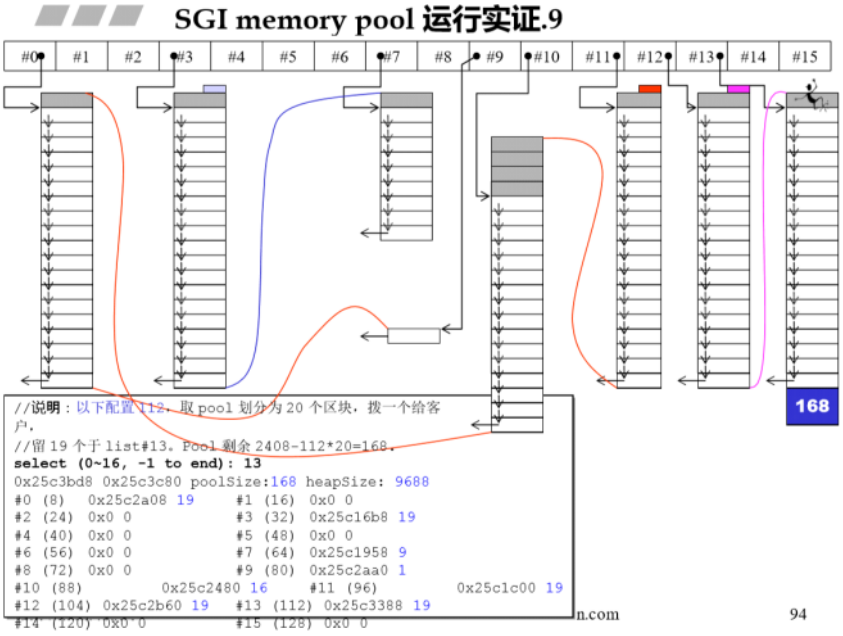
最多只切 20 个，所以剩下的会继续做为战备池。6 中剩下的240 减去8 * 20 剩余 80

8.配置104



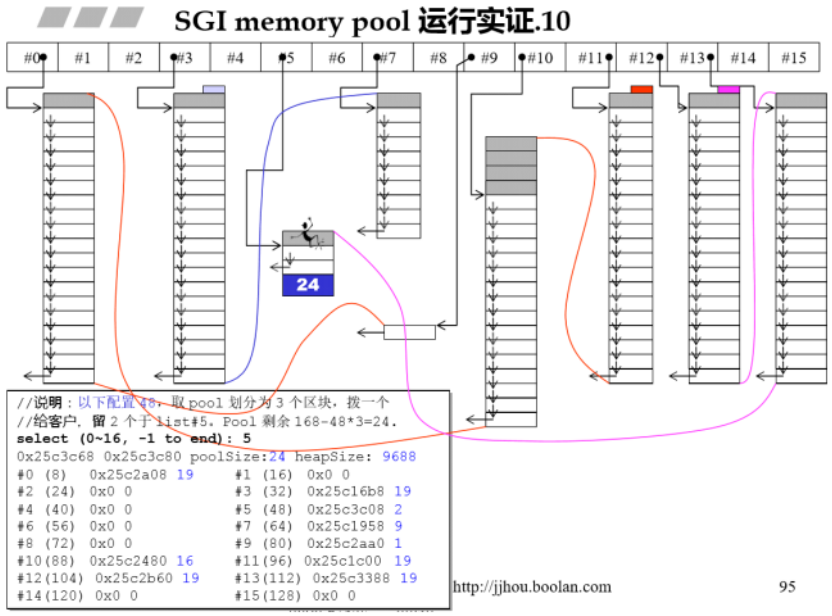
80 < 104, 一个也分配不出来， 所以挂到#9上/*#9正好是80的位置*/

9.配置112



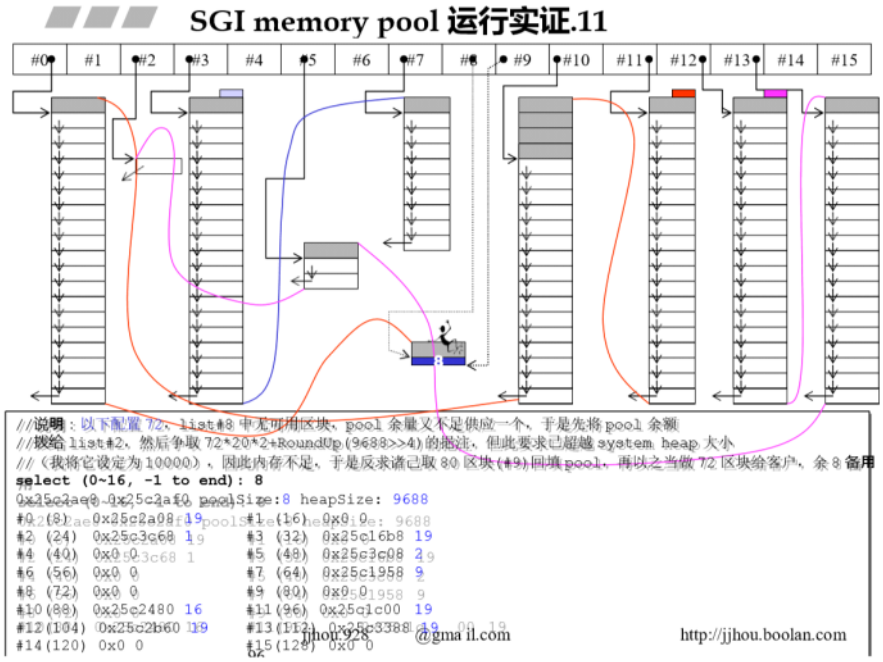
从2408中切

10.配置48个



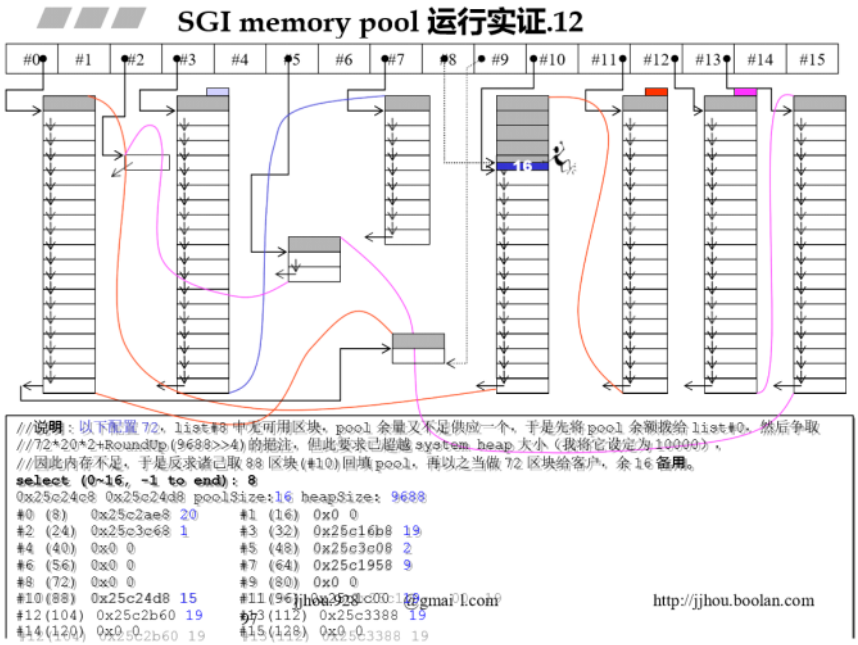
从168中切3个/*战备池足够就不多切了*/

11. 配置 72



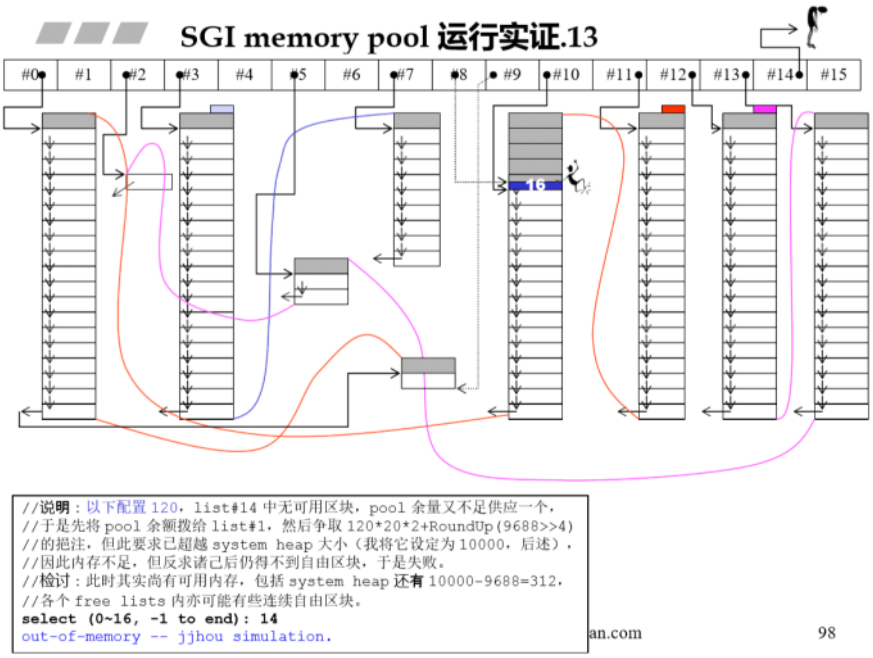
战备池一个也切不出来。 先把 24 挂到#2上。 又因为（假设）系统内存最大为 10000， 则不够再分配 20 * 72. 于是从 #8 向右找有没有空的， 因为有#9 有memory pool,所以从#9分配， #9 现在是空。

12.



先把 8byte挂到#0， 继续向右找到#10。

13.



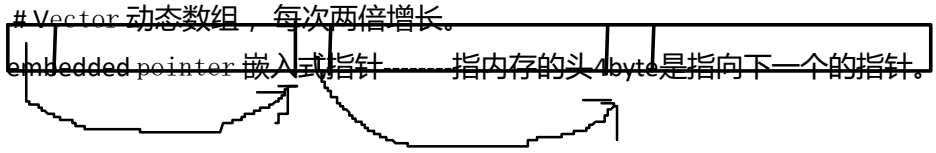
向右没有了，无法再分配。（向左会很麻烦）

容器的push_bakc 是拷贝一份过去。 无论外部元素是stack 还是heap， 拷贝到容器中后，都是拷贝到容器的内存中，容器中的内存是alloc出来的（从heap）

++

Vector 动态数组， 每次两倍增长。

embedded pointer 嵌入式指针------指内存的头4byte是指向下一个的指针。



+++

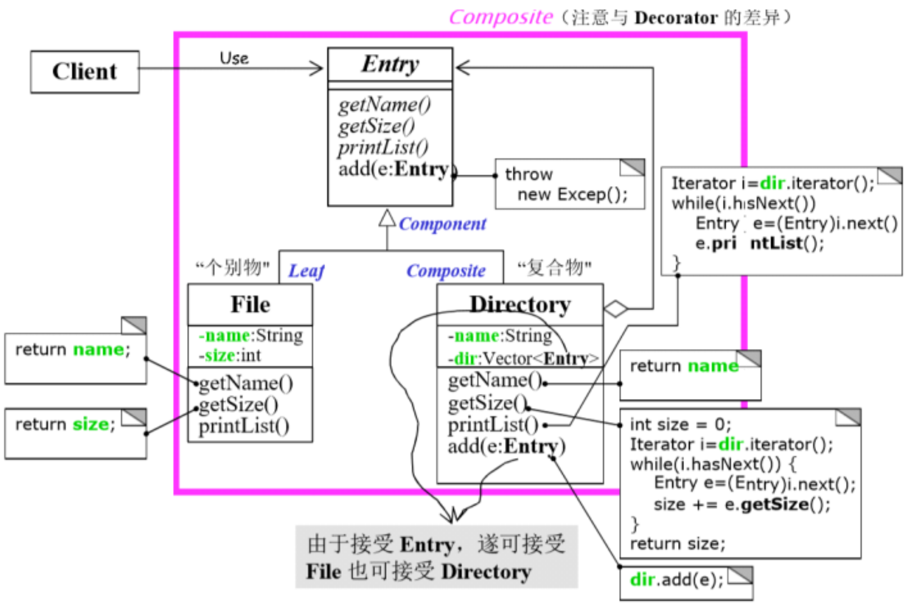
static void (*set_malloc_handler(void (*f)()))() ;

\\ typedef void (*H)();

\\ H set_malloc_handler(H f);

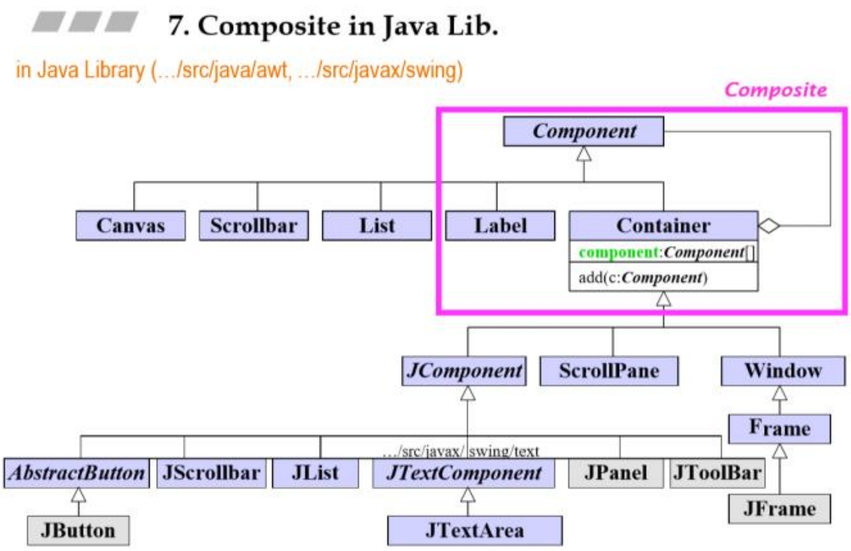
4. Composite

将对象(s) 组成/构成 为树状结构，用以表示 "局部-全部" 阶层体系。 Composite 可以让 clients 以一致的方式对待「个别对象」和「合成对象」。



File和Directory都继承自Entry,而Entry作为被统一对待的东西。并且会出现递归(getSize()方法)。

Add()函数在父类有定义的原因：如果父类没有写，子类写了add()，那么就需要判断某Entrys是什么。因此这里写在父类并抛出异常，右边会重载，左边会抛出异常。（经典做法）



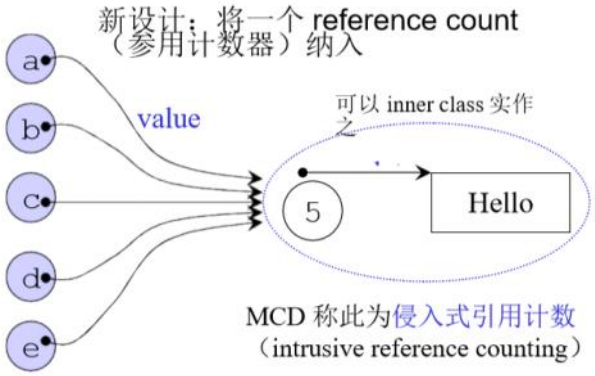
5.Reference Counting in MEC

只要有指针 --- 拷贝赋值（深度复制），拷贝构造，析构

非侵入式：

C++ 2.0 shared_ptr<T> “外套”

侵入式引用计数MCD：



实现见课件 P50

****

相同 class 的各个 object 互为友元

const obj 只能调用 const 方法

non - const 可以调用 const 和 非 const 方法

当一个方法的const 和 non-const方法都存在时，const obj 调用 const方法， non-const 调用 non-const方法

****

6. Adapter in GOF

转换 class 的接口使其为 client 所期望。Adapter 使得原本因「接口不兼容」 而无法合作的 classes 变得可以合作。

7. Proxy in GOF

为对象提供一个代理人或占位符号，用以控制对该对象的存取

```
class CharProxy {     // proxies for string chars
public:
  CharProxy(String& str, int index);              //建构
1 CharProxy& operator=(const CharProxy& rhs);     //左值运用
2 CharProxy& operator=(char c);                   //左值运用
  operator char() const;                          //右值运用。转换函式
private:
  String& theString;          //proxy 所附身（相应）之字符串。
  int charIndex;              //proxy 代表之（字符串内的）字
};
```

C++较罕见：使用 reference variable

转换函数：：把灰色的CharProxy 转换为char ---->

格式： operator[ ][returnType]() [const];

//operator + 空格 + 被转换成的类型 + const

//当是右值时自动调用

//operator=只有在proxy中重载

8.Command, in GoF