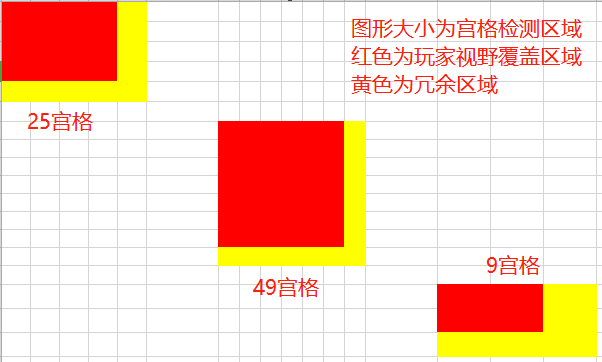
**关于选择25宫格**

首先虽然常规的都是9宫处理，但是不代表9宫是最优的，就像寻路，常规都是A\*处理，不代表A\*就是最优的。其次，AOI需要满足的是玩家周围半径内(或摄像机内)n米内的单位可见(这里可再做优先级筛选)，每个AOI格的大小，是我们可调整的。可分为3个档位9宫格、25宫格、49宫格，而此时，如果用9宫，则每宫的边长至少为n，而25宫是n/2，49宫是n/3。然后，如果单位密度全一致的情况下，9宫格的冗余视野面积为5n^2(5平米)，而25宫格的冗余视野面积为9/4n^2 (2.25平)，49宫格的为13/9n^2(1.44平)，25宫格比9宫格冗余部分少了55%，计算量(格子遍历数+177%)，但是单位遍历数却少了(因为面积小了)，大头都在遍历到单位后的判定，而不是取到Tower的开销，统计频率增加1倍(大约AOI上2倍CPU开销)，49宫格比9宫格冗余部分少了71.2%，49宫格的计算频率是9宫格的3倍。也可以算是3倍CPU开销，但是相比25宫格，多了一倍的CPU开销，只换来16.2%的冗余面积减少，比25宫格多1倍开销，换55%的减少，性价比少了2.39倍。宫数越多越亏，那为什么我们要用25宫呢，因为我们CPU有富裕，而带宽没有，我们已经过了可劲儿优化CPU的时期，现在是可劲儿优化带宽的时期，所以，能用CPU换带宽的事儿，只要性价比不低，我们就换。所以选择25宫格是性价比最高的。



**合批策略(基于TCP Nagle算法)**

16ms是测算的一个合理值，这块儿，无非几种做法，要么来自客户端的每个request做累计(处理过程中产生的所有推送累计)，（并且每次场景tick累计），要么时间累计（框架使用），两方案都可其实，我上层业务(目前先说客户端消息)的处理间隔是5ms,场景tick间隔是30ms，而且咱们存在多个线程同时操作channel的情况，故最终选用了后者，10-20ms，都差不多，但是这个值越小，缓存的量越少，这个值越大，延迟越高。所以后来选择了16ms，大约为场景tick的一半。因为30ms是接受不了的。这块儿未来可以有优化空间，就是这两个方案结合。就是来自客户端的消息，当前帧处理完，立刻给与返回，这样服务器逻辑端，相当于只有0-5 ms延迟。而不是现在的0-21的延迟。

**从ItemData了解池化**

物品的析构有几种，一个是在背包中消化，就是被使用/堆叠等等行为，此时会进行池化回收，另一类，就是传输过程中丢失，如邮件，或者到了中心服，进行商城等行为。首先，我们不做跨线程的线程池，然后每个 接收口，在对应线程做池。比如线程A把数据丢给线程B，那么线程B用完了，就回收到线程B的池里，接下来线程B再用的时候，就能快一点。那你可能会有疑问，要是一直A给B丢怎么办，这种情况属于极限情况，通常情况下，是等权重的。池化的存在，只是为了减少一些gc，并且让再次创建对象的时候快一点，池不启动项目照样跑。所以我们的池，就是尽量的回收一些对象，不一定是全部(即使某一处接收口忘了写，也只是影响这个对象的利用率而已)。另外，我们的池也都是有上限的，超上限就弃了，目前上限是手动设定，未来有空了把自动池做出来。所谓自动池，就是达到额定上限后，统计单位时间内的 冲击数，如果冲击数超过池上限的n%，就扩容来解决我们设计初期，可能导致的预估失误。

**关于战斗逻辑中的group**

战斗系统中，unit,skill,buff,attack,cd都有这个设定，group的含义基本都一致，就是给某个实体贴上标签。比如，单位组，可以有地面单位组、空中单位组、进战组、物理伤害组、高防御组、法师组等等。可以按策划意图从各个去维度划分，攻击也可以如此，物理攻击、魔法攻击、冰系、火系、无伤害系。buff也可如此，增益、减益、属性增强类、控制类、中毒类以及特定类。这些group，程序不用关心，具体某个groupID是几，只要使用者和被标记者，能对上就可以了。比如，我有一个buff，效果是所有冰系技能level+1，那么什么是冰系技能呢？ buff里配置一个groupID，然后所有skill里，包含这个group的，就视为是这个冰系技能，每个实体可以同时属于多个组。

**关于配置表中部分冗余字段的设计目的**

关于冗余的问题，万事无绝对。如果少量的冗余，能够带来优美的结构，良好的体验，强劲的性能，那这种冗余就是可以存在的。以物品表的使用限制等级和使用条件两个字段为例，物品使用时，都要判断使用等级。而RoleConditionType里也有角色等级这一条。在不影响逻辑和性能的前提下，配置还是要考虑的稍微人性化一点。所以常用的点，就会独立出来，不常用的，自然就不用独立，就放通用condition里就可以了，而且还好扩展。例如quest表的前置任务组和接取条件两个字段为例，RoleConditionType也有任务完成这一条，看似跟前置任务的作用一样。但是preQuests是带逻辑的，通过这个来反查出后置任务。这样每次，完成一个任务之后，不用遍历全部任务表，只需遍历后置任务组即可。所以这两个东西需要分开。你可以认为，RoleCondition里的QuestComplete就不是给任务模块准备的，虽然用了也会生效。