ceph 作为分布式存储系统，采用多节点多副本的数据存放方式，必然要解决数据如何分布到集群中各个节点和磁盘上。本节内容就是介绍ceph的crush数据分布算法。

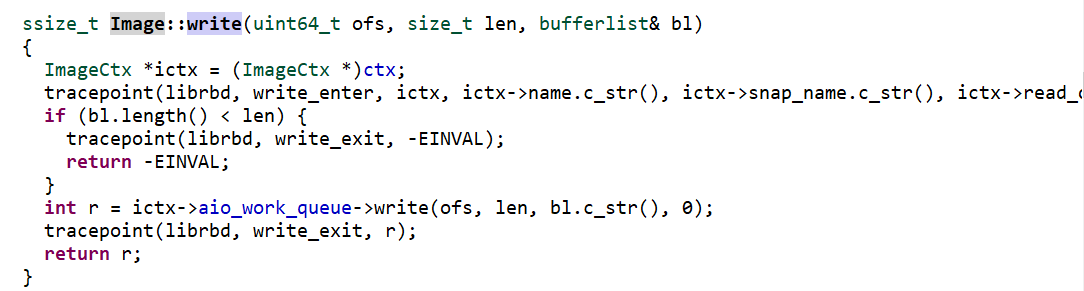
例如一个ceph集群三副本，就存在着如何映射3个OSD存储这3个副本的数据，ceph写数据时，即写object时，首先需要计算出object属于哪个PG，然后根据PG id 计算出存放的OSD位置：

过程分两步：1. PG id的计算 2. OSD位置的计算

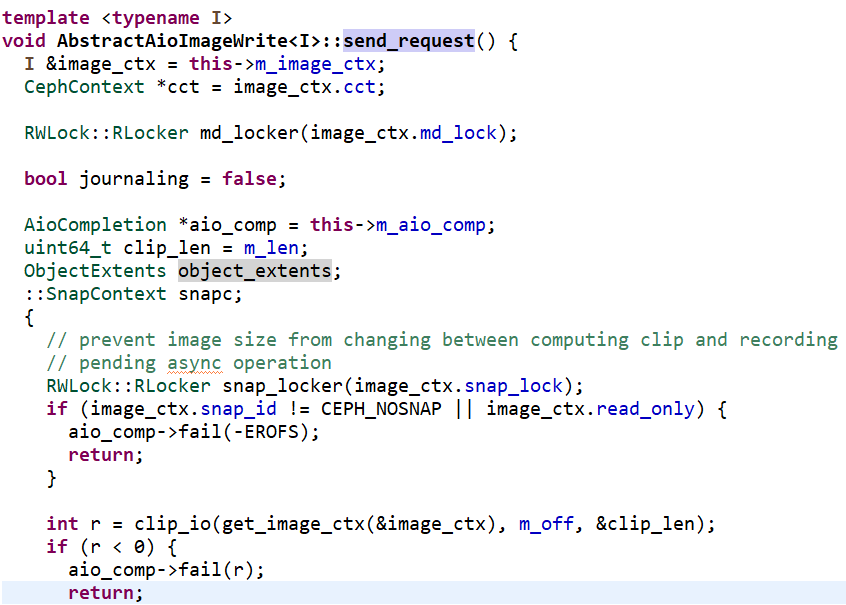
以rbd为例，结合代码介绍这两个过程：

PG id的计算：

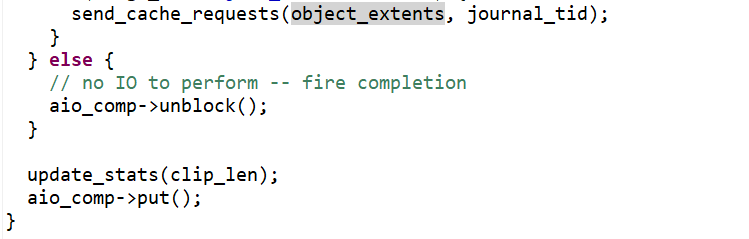
rbd的写接口



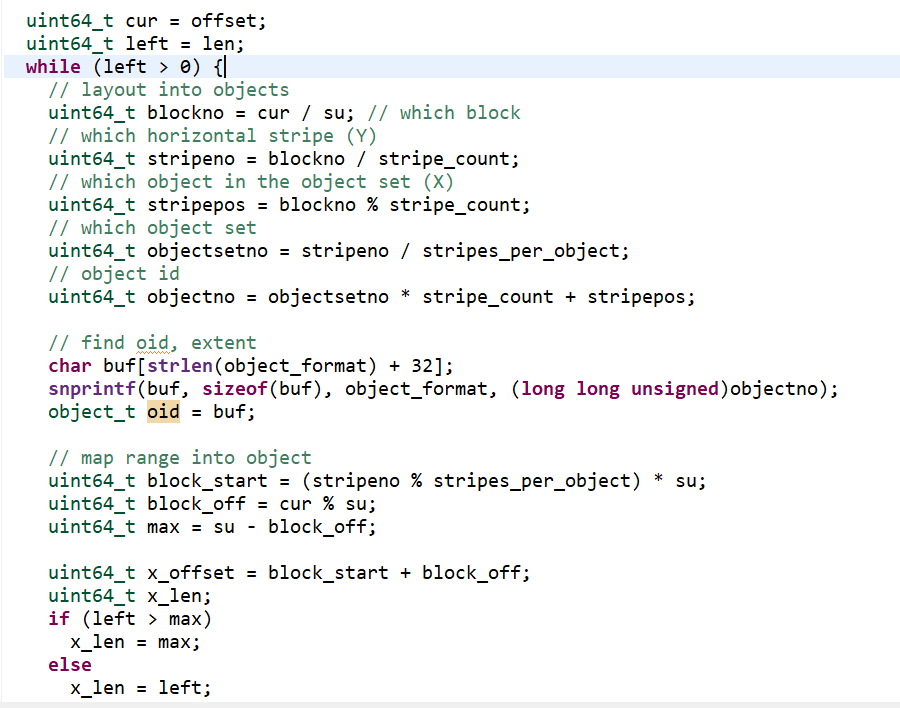
接口传入的参数是起始写位置（ofs）以及写数据大小（len）和要写入的数据（bl）,然后这段写的数据要进行切分，分成大小同等（可设定，一般为4M）的object(最后一块object可能大小小于块大小)。



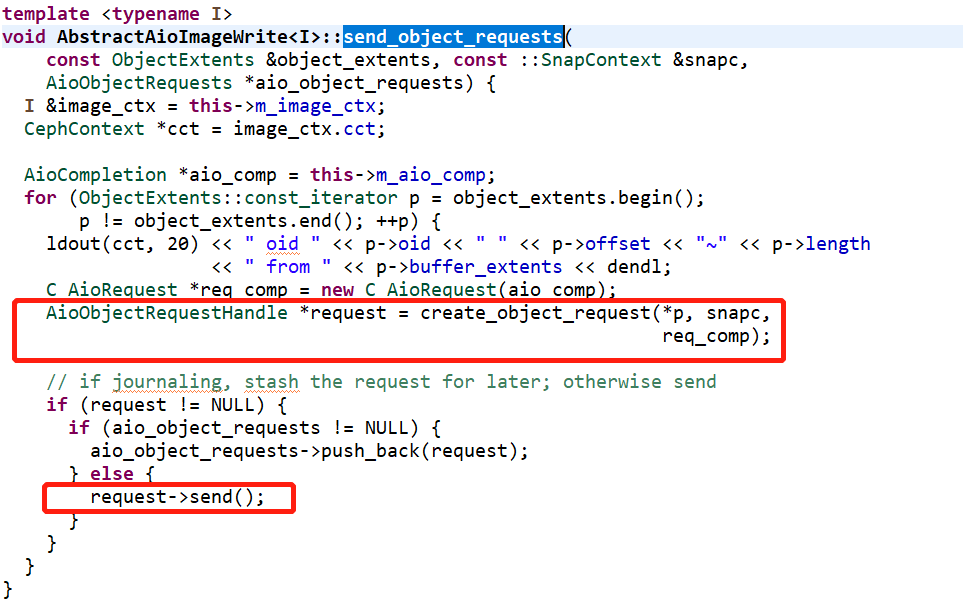


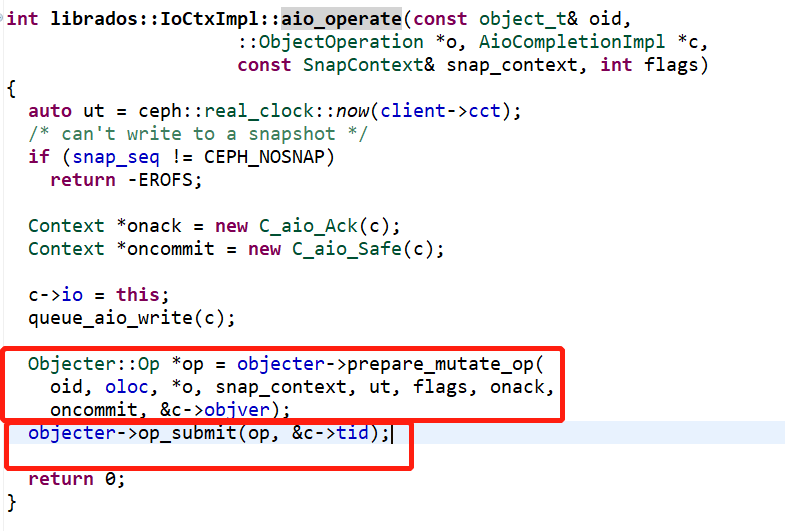


file\_to\_extents就是将数据段切分各个object：

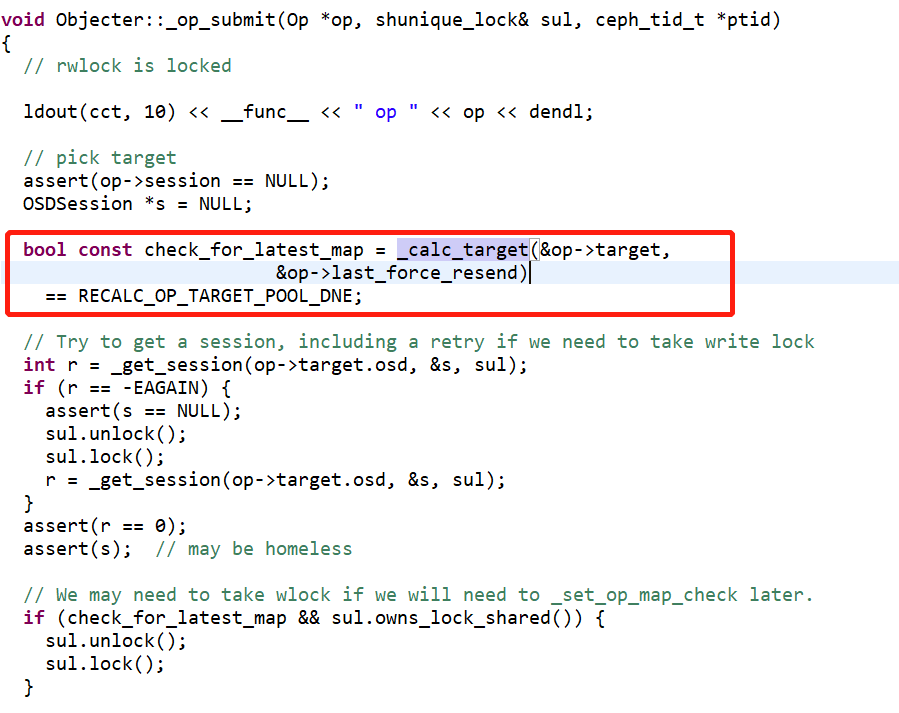


然后各个object分别构造写请求：

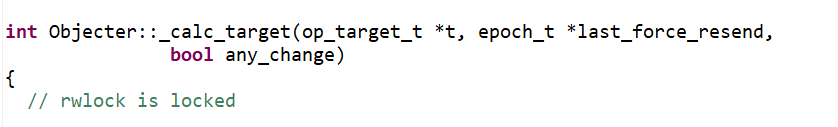


此后会构造objecter的Op请求，发送出去：  


在\_op\_submit函数中，\_calc\_target()就是计算当前object保存的OSD：



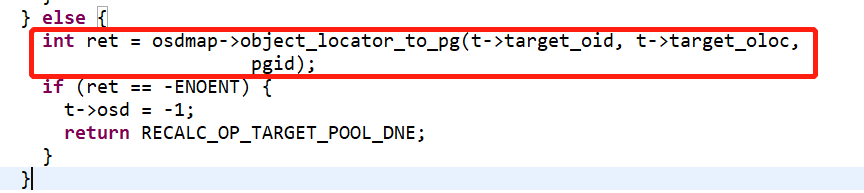
我们重点详细分析下\_calc\_target函数：



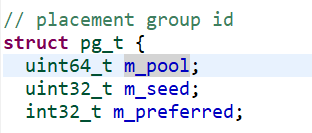
首先获取pool信息，判断是否有效：

const pg\_pool\_t \*pi = osdmap->get\_pg\_pool(t->base\_oloc.pool);

然后根据获取pgid,注意pgid是一个结构体pg\_t

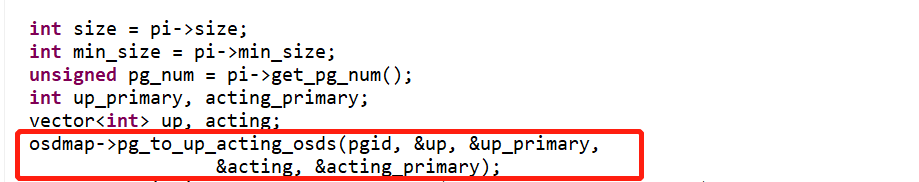


pg\_t 的结构如下：

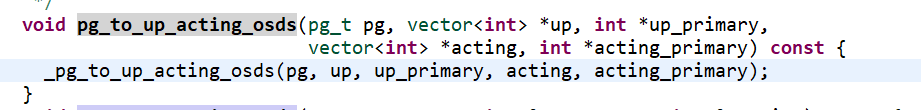


m\_pool 是pool id， m\_seed是函数根据object id算出来的哈希值，m\_preferred赋值-1。

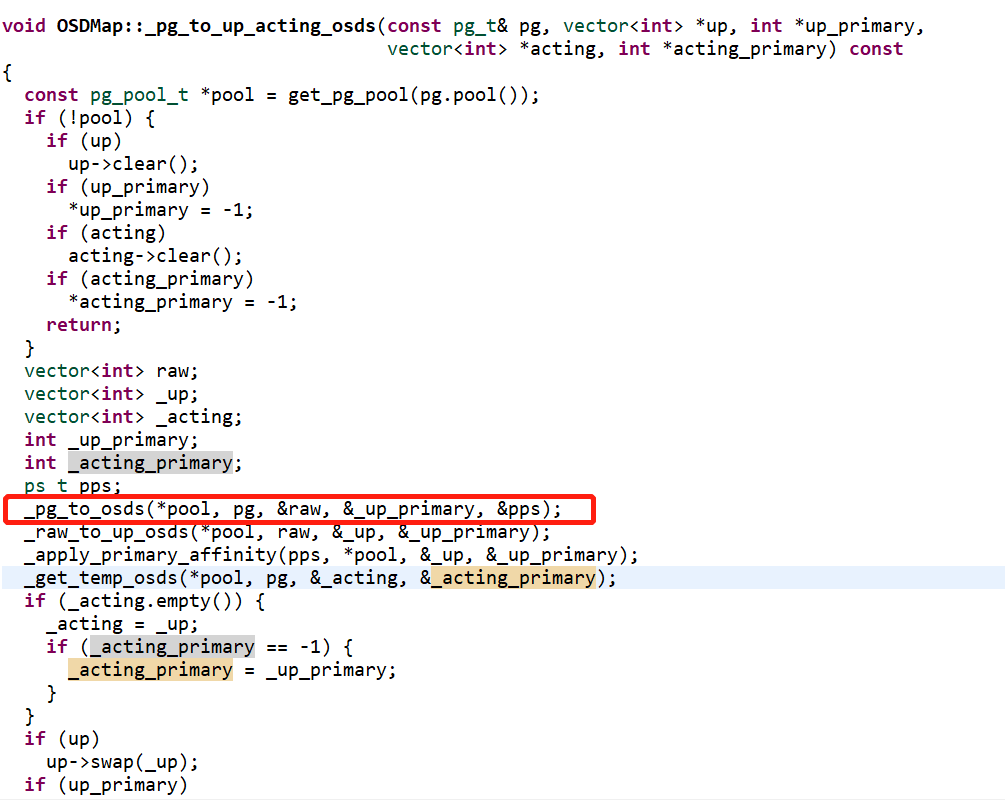
接下来就是选择osd了：



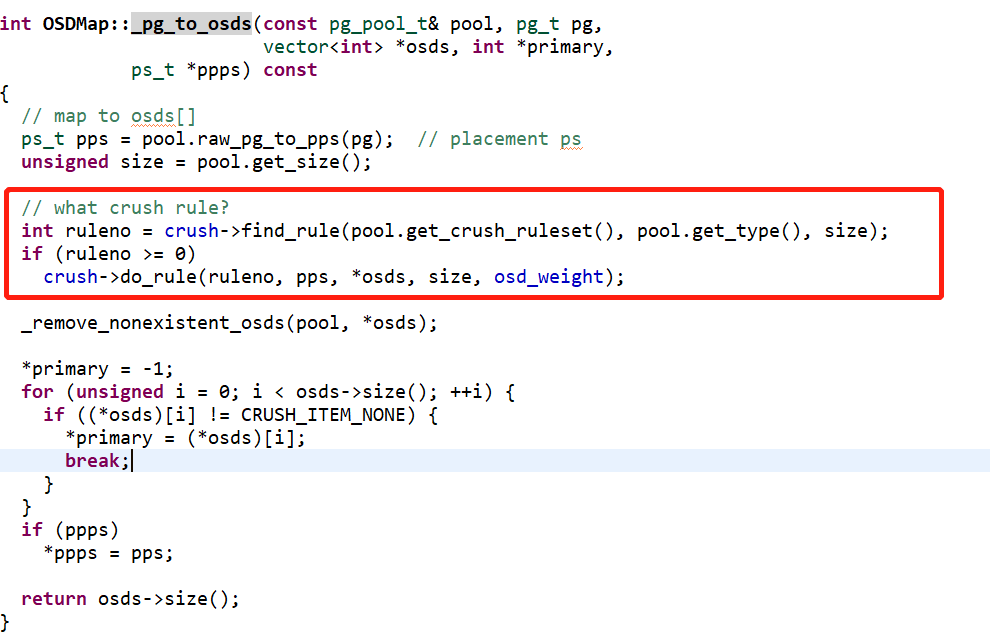
这个函数就是选出up osds以及 acting osds, 两个都是数组类型，大小为副本数



继续跟踪这个函数：

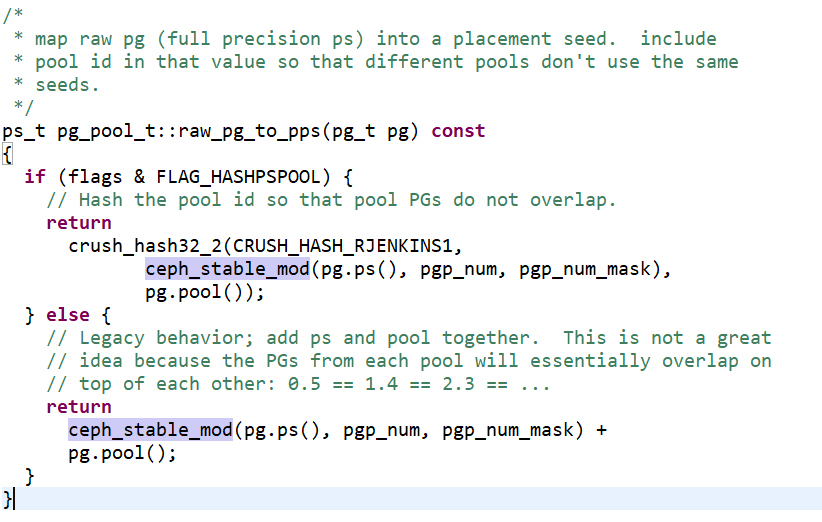


进入\_pg\_to\_osds：

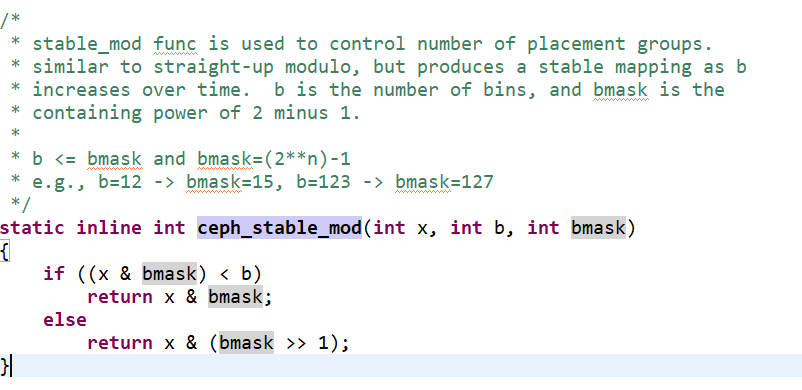


上面函数crush->do\_rule()就是真正调用crush算法计算出相应的osd列表。这里重点解释下参数pps:

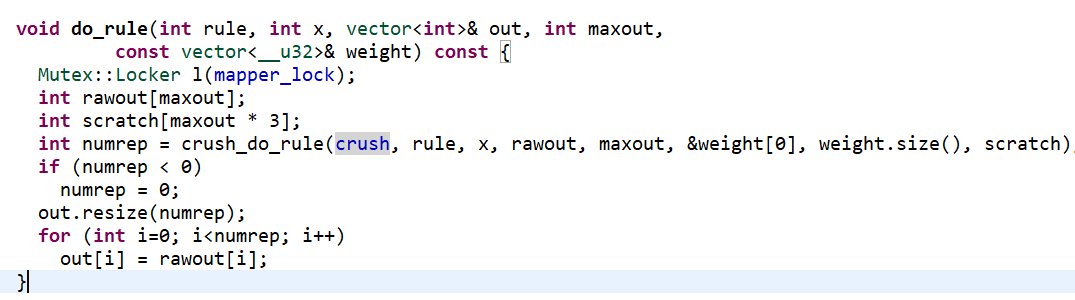
ps\_t pps = pool.raw\_pg\_to\_pps(pg); // placement ps

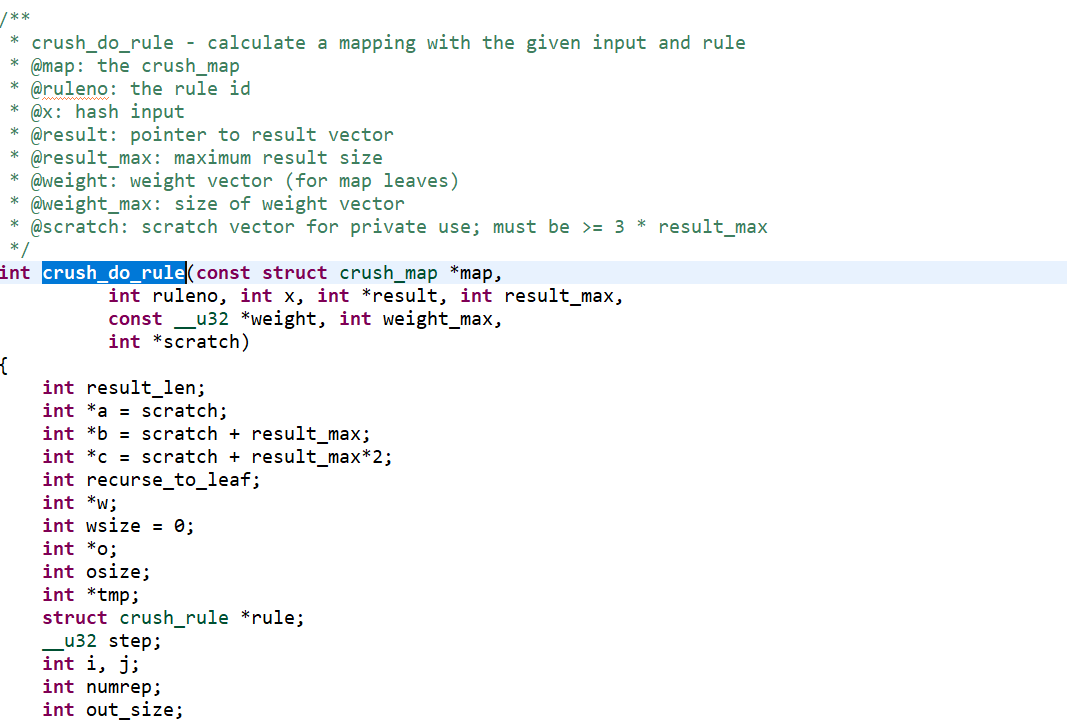


可以看出pps这是一个哈希值，这个哈希值根据pool id，函数中pg.ps()就是我们object哈希算出的m\_seed：



下面就是进入do\_rule 进行crush分布式算法的处理了：





我们详细分析下crush 算法：

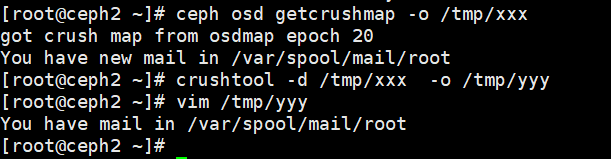
crush 算法输入需要3个输入参数：

输入x 即PG id的哈希值

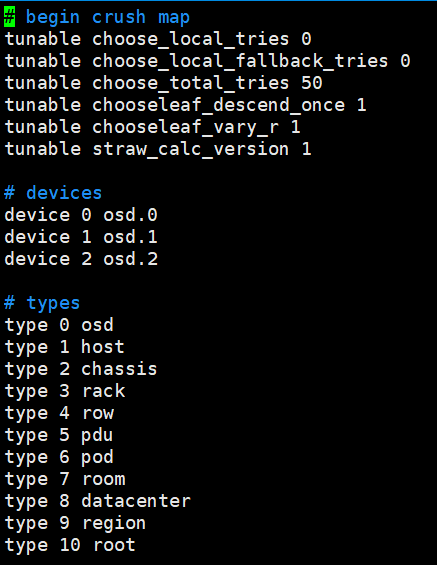
crush\_map即集群的拓扑结构

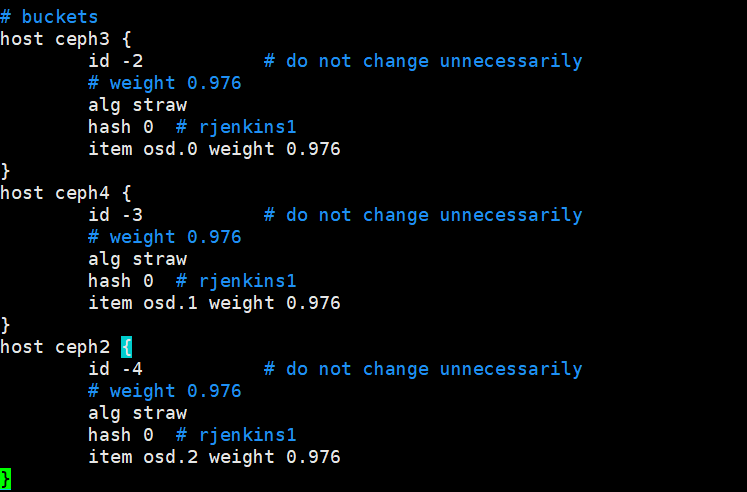
ruleno 即选择策略，就rule规则，这里用编号表示

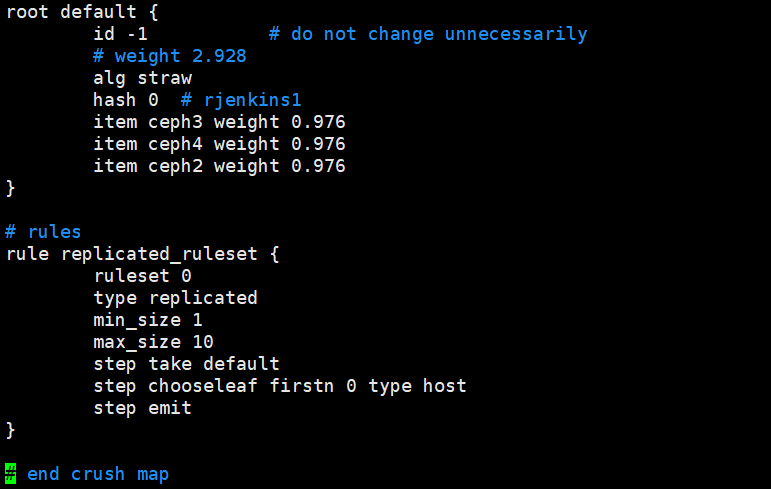
我们先看看crush\_map, 可以通过集群输出crush\_map:



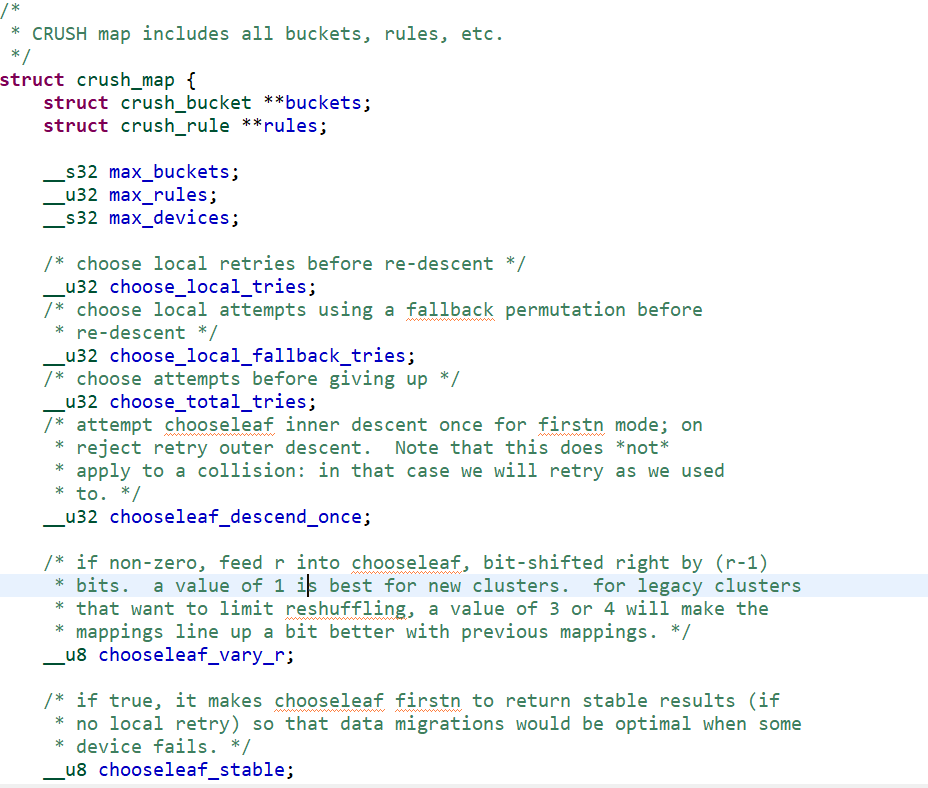
vim crush map如下：



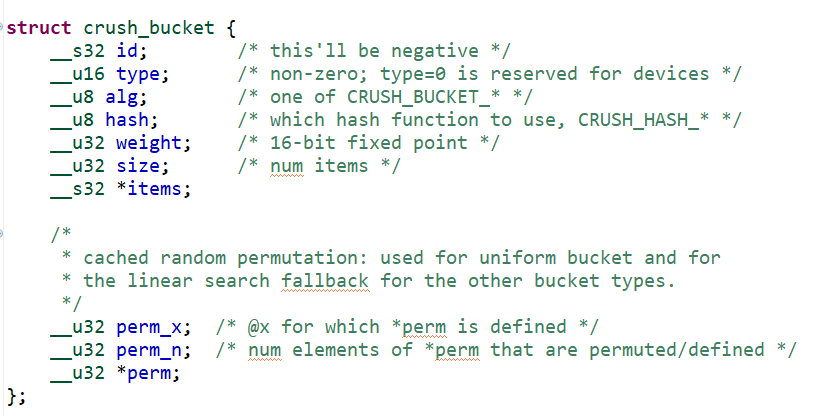




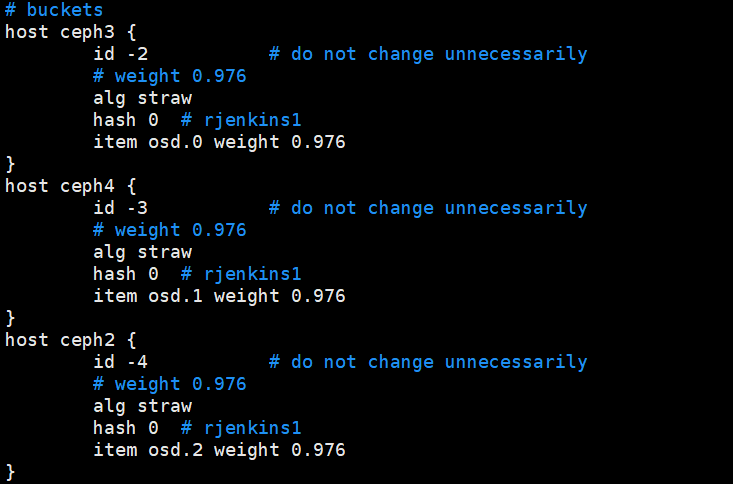
显示的结构和代码中的结构还是有着映射的关系：

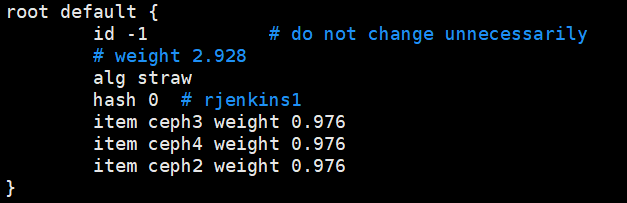


其中crush\_bucket:

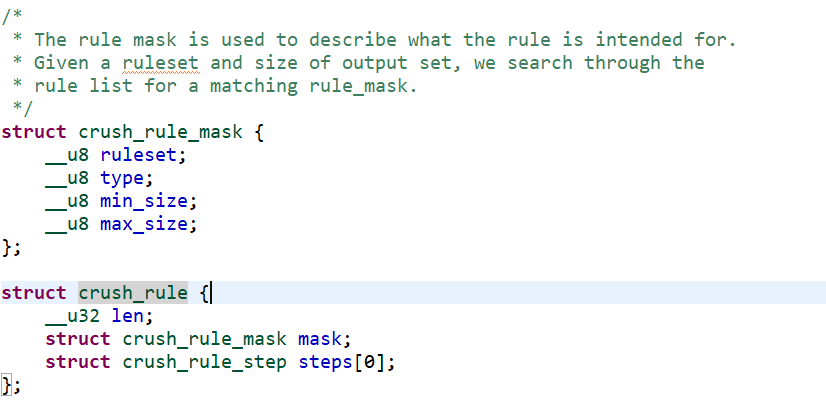


对应：

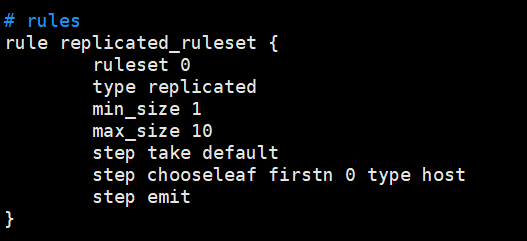




crush\_rule:



对应于：



大家可以逐一的对比分析其数据结构，这里不做展开。

这里分析下其选择OSD的过程：

**int** \*a = scratch;

**int** \*b = scratch + result\_max;

**int** \*c = scratch + result\_max\*2;

w = a;

o = b;

a, b, c 分别指向 scratch向量的0, 1/3, 2/3的位置.   
w = a; o = b;   
- w被用作一个先入先出队列来在CRUSH map中进行横向优先搜索(BFS traversal).   
- o存储crush\_choose\_firstn选择的结果.   
- c存储最终的OSD选择结果.

crush\_choose\_firstn计算后如果结果不是OSD类型, o 交给w.以便于 w成为下次crush\_choose\_firstn的输入参数.

crush\_do\_rule函数里面最重要的是函数里面的for循环，这个循环就是筛选osd的过程，

for循环中：

首先从rule规则中当前执行的步骤，首次就执行第一条步骤：

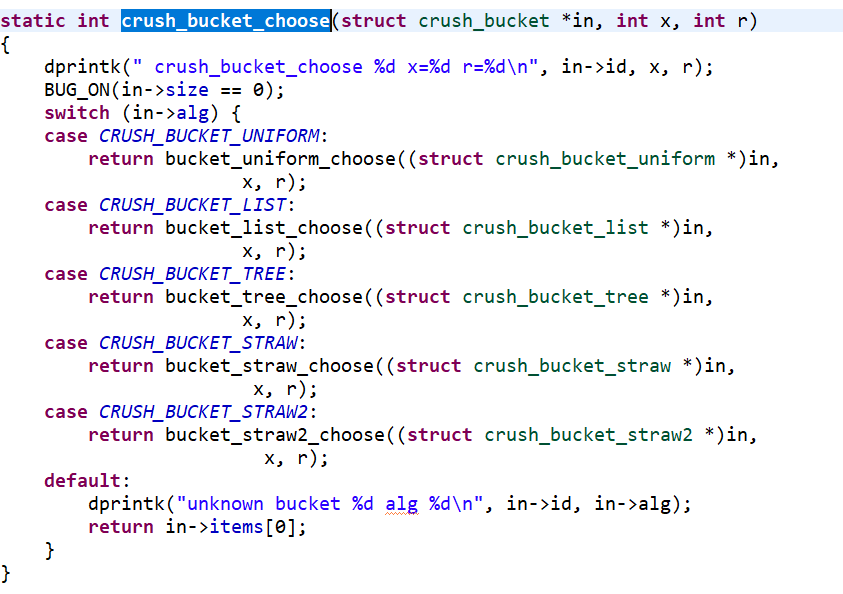
struct crush\_rule\_step \*curstep = &rule->steps[step];

然后根据当前执行步骤的操作类型，选择不同的分支操作，首先一般是take操作，而且是take fault。即crush map树根节点。这个过程就是根据step 逐步选择bucket 知道知道叶子节点，即OSD。

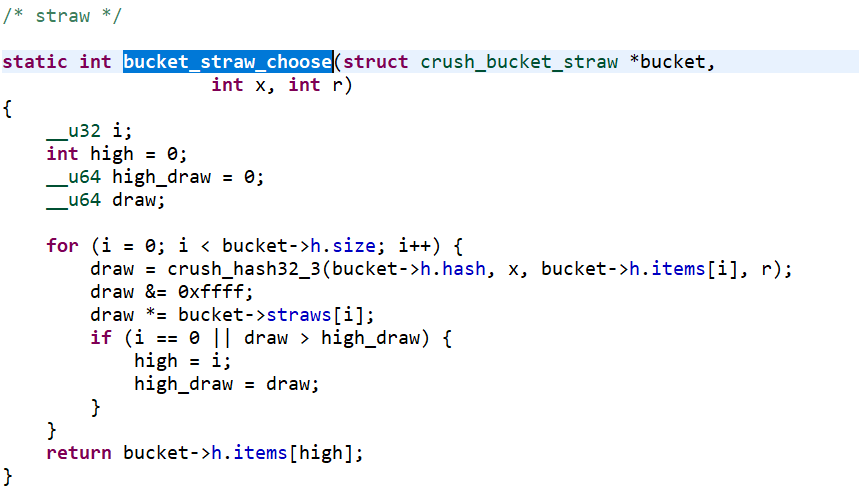
这个过程中，crush\_choose\_firstn 函数

这个函数递归的选择特定bucket或者设备,并且可以处理冲突,失败的情况.   
如果当前是choose过程,通过调用crush\_bucket\_choose来直接选择.   
如果当前是chooseleaf选择叶子节点的过程,该函数将递归直到得到叶子节点.

在这个函数中有crush\_bucket\_choose函数，这个函数根据bucket类型选择不同的权重计算方法刷选出bucket。我们采用straw，就会采用bucket\_tra\_choose接口进行刷选。



其通过计算节点权重，以权重最高的作为选择出的节点



至此，crush map 算法大致如此。

其就是通过构造crush map，类似一颗树，从根节点按照设计的step方案，依次循环遍历，递归，刷选出符合要求的叶子节点即OSD，选出符合副本数的OSD，就刷选完毕。