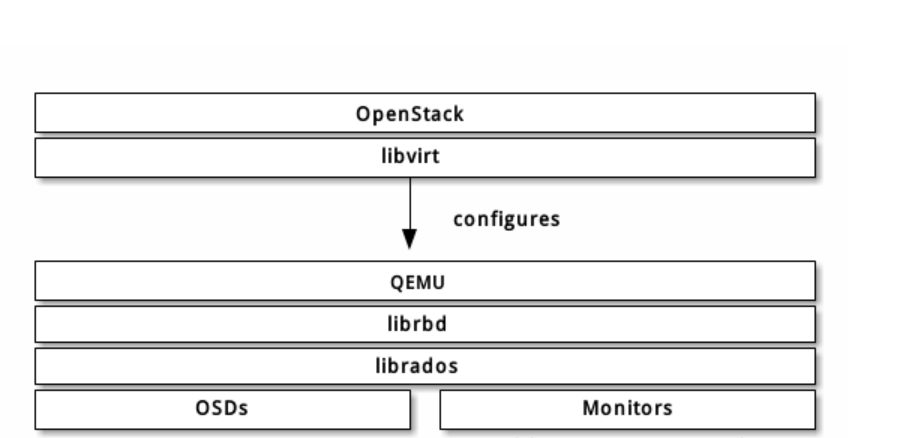
提供块存储是ceph非常重要的一个功能，RBD是ceph提供的块设备，本章节主要介绍其rbd客户端实现流程。

openstack对接ceph，其rbd数据流如下

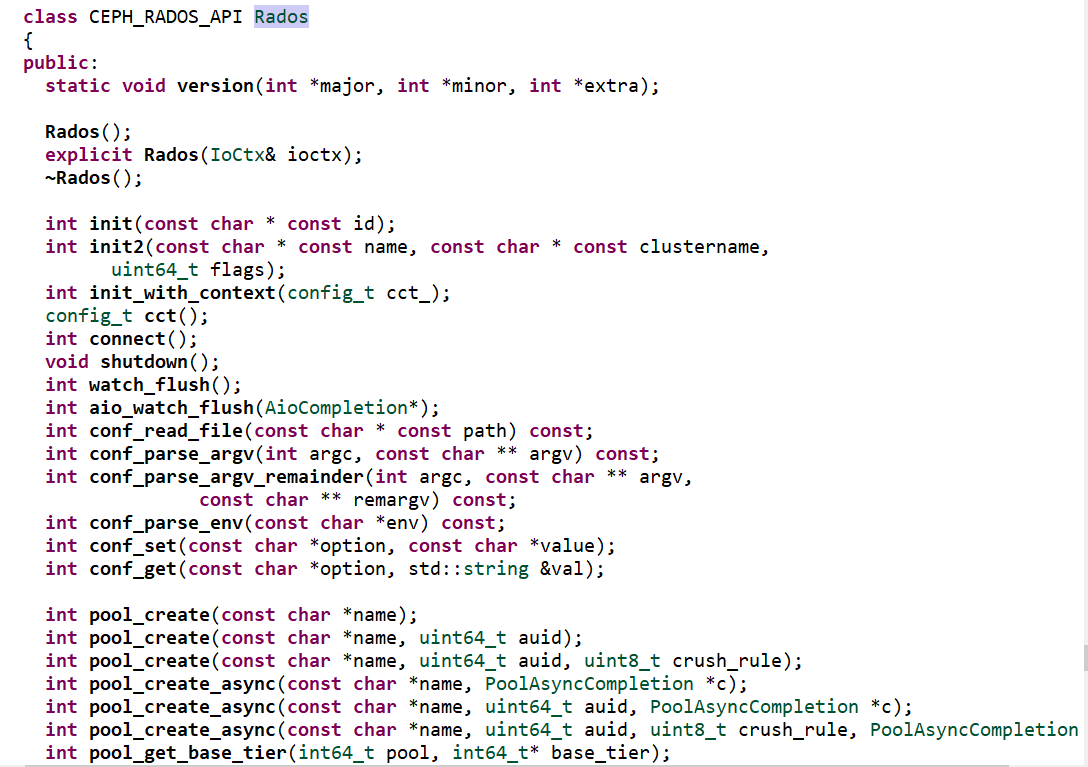


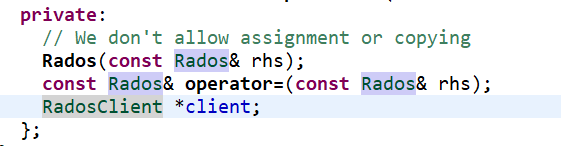
其中librbd以及librados都是属于ceph 的客户端，其提供ceph的接口向上提供块存储服务。

LIBRADOS提供客户端访问Ceph集群的原生态统一接口。其它接口或者命令行工具都基于该动态库实现。在librados中实现了Crush算法和网络通信等公共功能，数据请求操作在librados计算完成后可以直接与对应的OSD交互进行数据传输。LIBRBD 是CEPH提供的在librados上封装的块存储接口的抽象。

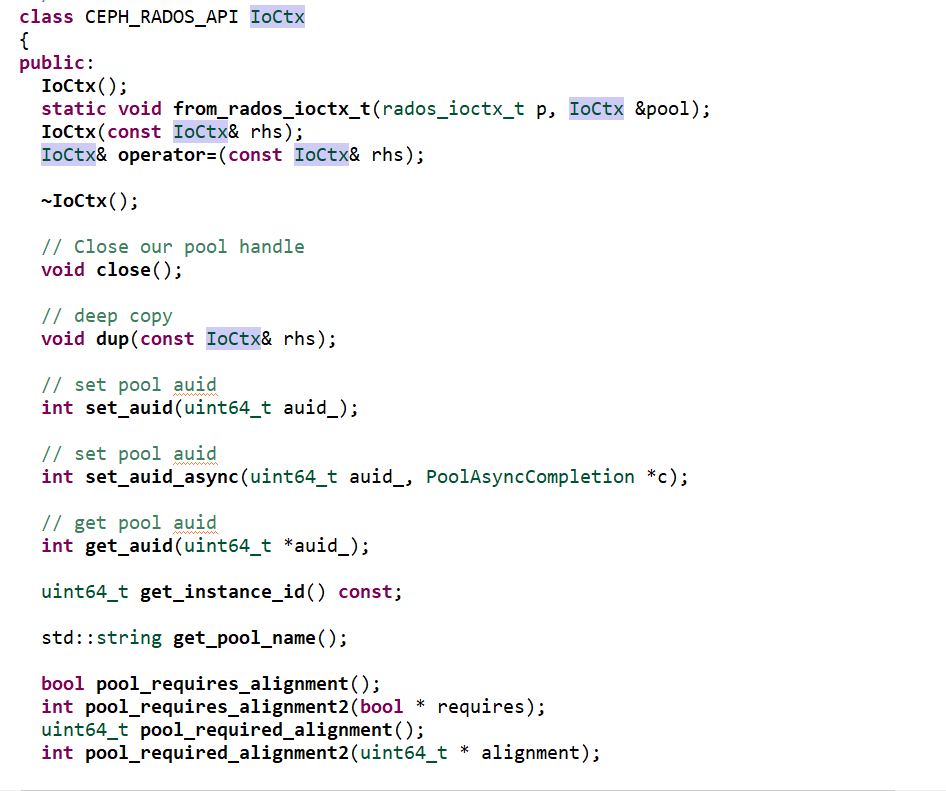
librados主要的类是

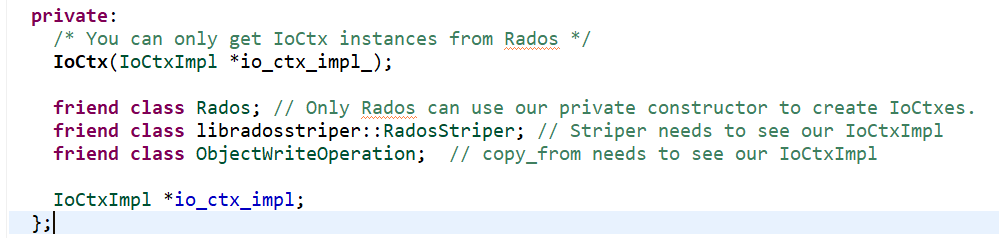
librados::Rados    负责初始化集群、读取配置、连接集群





librados::IoCtx    负责创建IO上下文环境



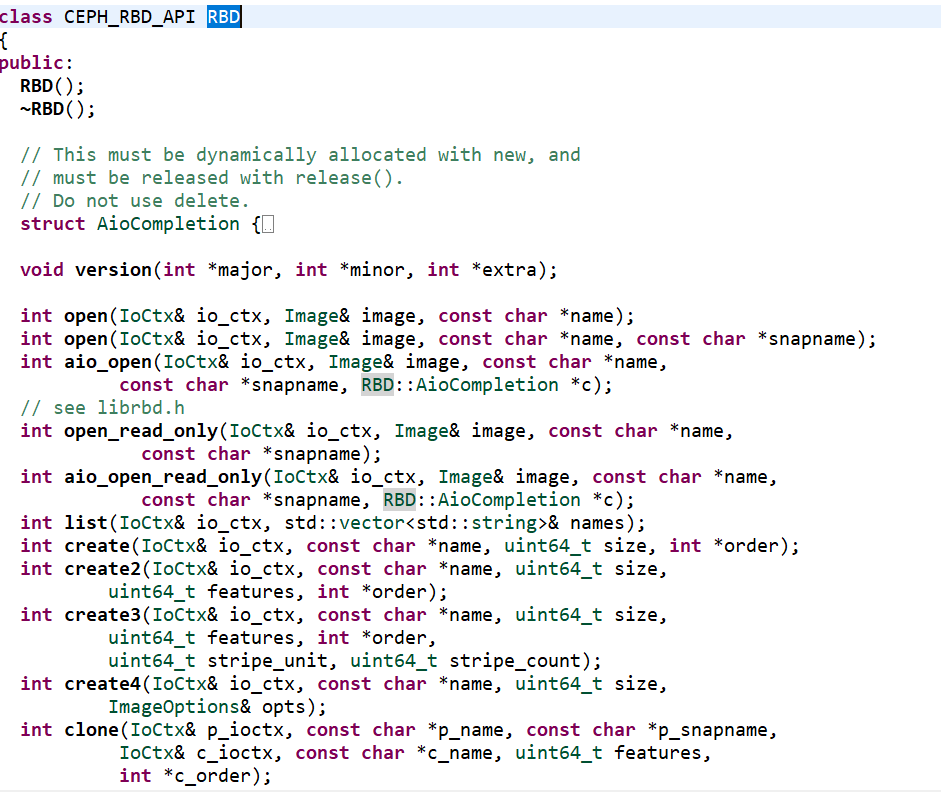


librados::bufferlist  负责读写缓存

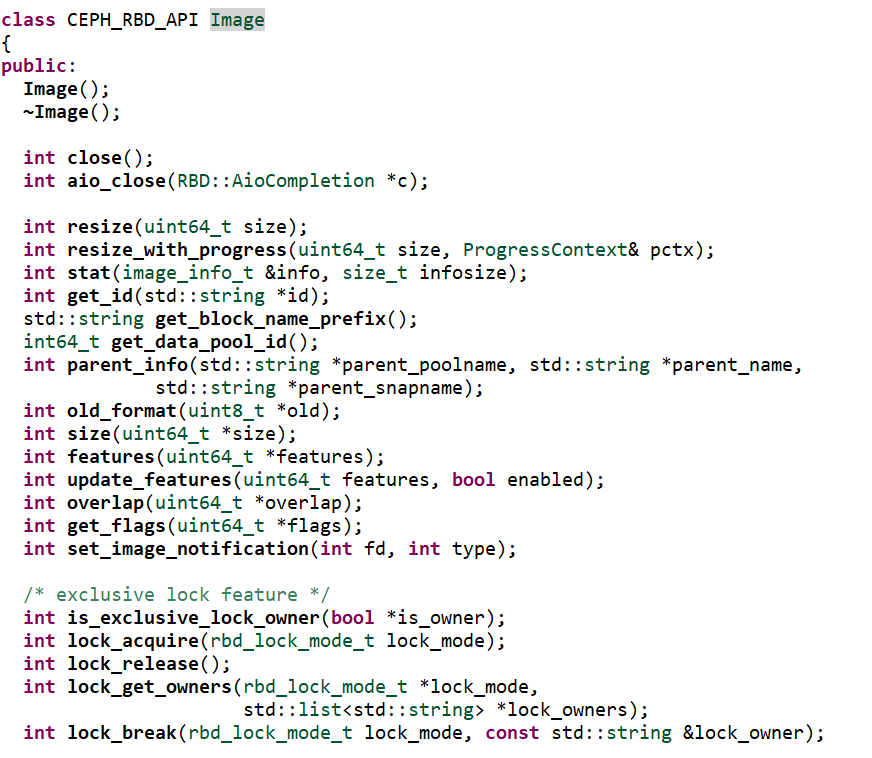


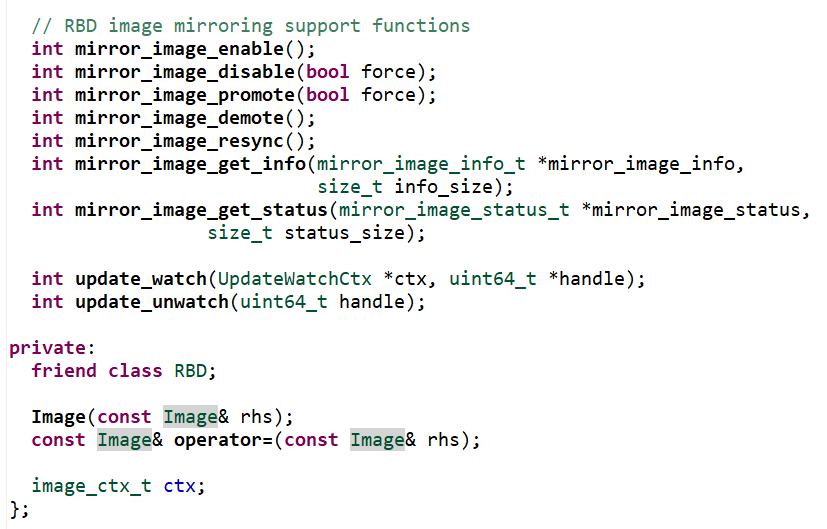
librbd最主要的两个类是

librbd::rbd  主要负责创建、删除、克隆映像等操作，



librbd::image  负责映像的读写等操作





对于任何客户端的读写都要经过以下步骤：

1. 集群句柄创建、读取配置

集群句柄的创建即是librados:Rados的创建，初始化，读取配置

创建：librados::Rados rados;

初始化：librados::Rados::init(const char \* const id)

主要是初始化librados::RadosClient

读取配置：

librados::Rados::conf\_read\_file(const char \* const path) const

librados::Rados::conf\_parse\_argv(int argc, const char \*\* argv) const

1. 集群连接

librados::Rados::connect()

1. IO上下文环境初始化（poll创建读写等）

librados::Rados::ioctx\_create(const char \*name, IoCtx &io)

主要是IoCtxImpl即librados::IoCtx

1. rbd创建

librbd::RBD rbd;

RBD::create2(IoCtx& io\_ctx, const char \*name, uint64\_t size,

uint64\_t features, int \*order)

1. rbd的读写

librbd::Image image;

RBD::open(IoCtx& io\_ctx, Image& image, const char \*name)

Image::write(uint64\_t ofs, size\_t len, bufferlist& bl)

Image::read(uint64\_t ofs, size\_t len, bufferlist& bl)

1. IO上下文环境关闭

librbd::Image::close()

librados::IoCtx::close()

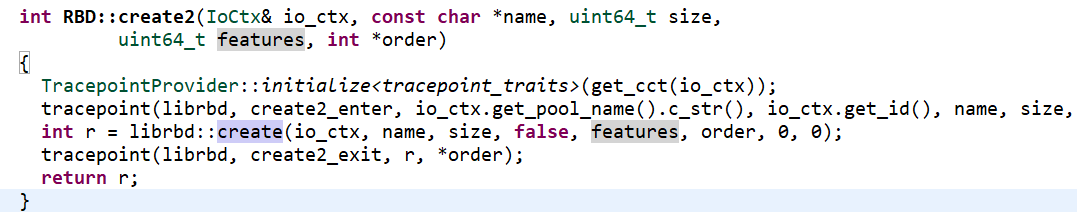
7. 集群句柄关闭

librados::Rados::shutdown()

我们集中讲解以下几个流程：

1. rbd卷的创建

rbd卷的创建接口：



函数输入参数：

io\_ctx: 针对pool的上下文环境，对pool的操作都要首先建立一个相应的上下文环境

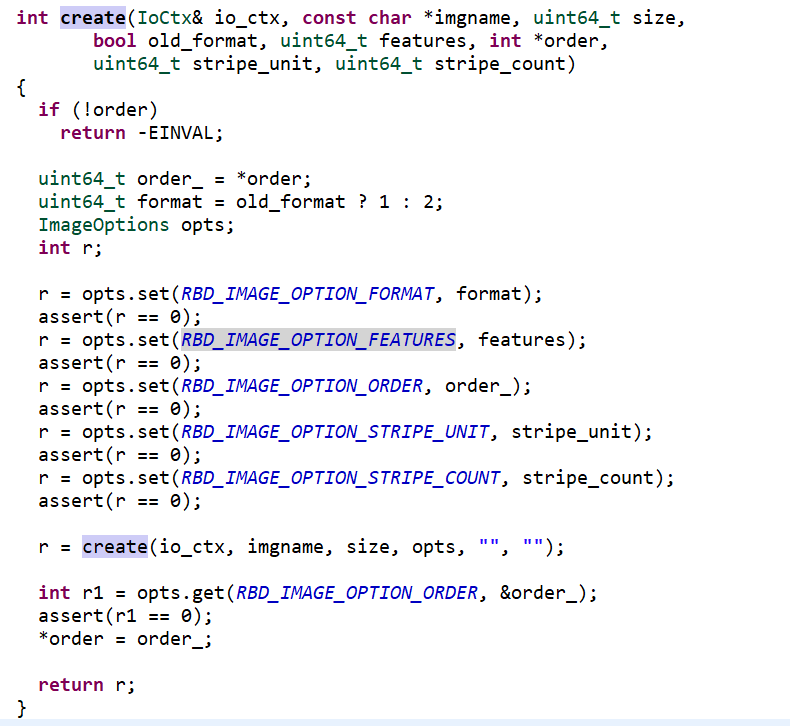
\*name：rbd卷名字

size：rbd卷大小

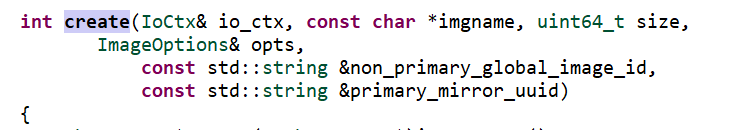
features: rbd卷的特性

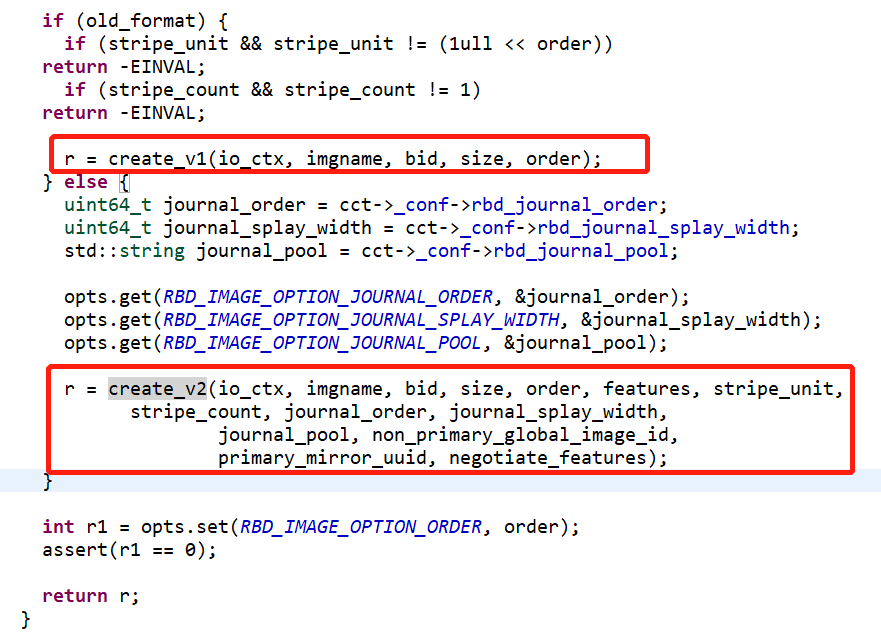
order: rbd卷的分块大小

其具体实现在internal.cc中：



继续往下调用：





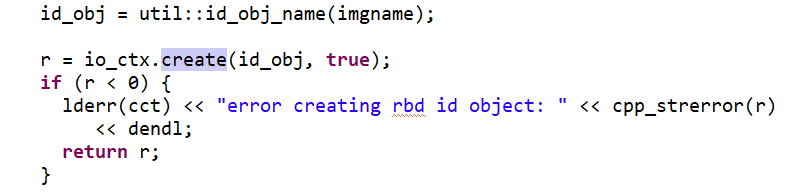
根据format格式调用不同的创建接口，我们采用新的format,即2，所用调用新的接口：

**int** **create\_v2**(IoCtx& io\_ctx, **const** **char** \*imgname, uint64\_t bid, uint64\_t size,**int** order, uint64\_t features, uint64\_t stripe\_unit,uint64\_t stripe\_count, uint8\_t journal\_order,uint8\_t journal\_splay\_width, **const** std::string &journal\_pool,**const** std::string &non\_primary\_global\_image\_id,

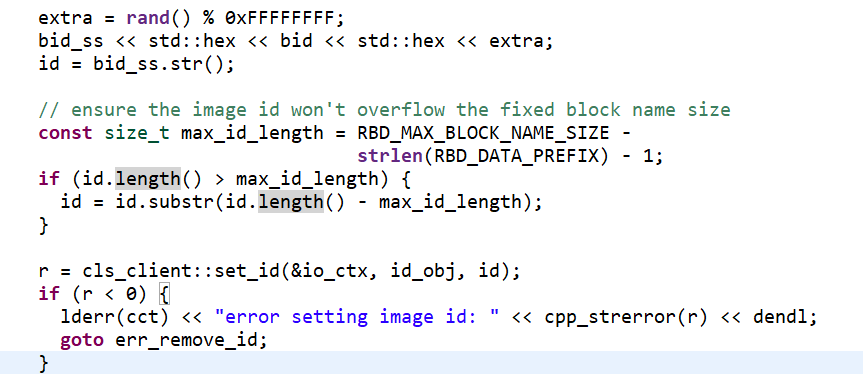
**const** std::string &primary\_mirror\_uuid,**bool** negotiate\_features)

这个接口会做如下工作：

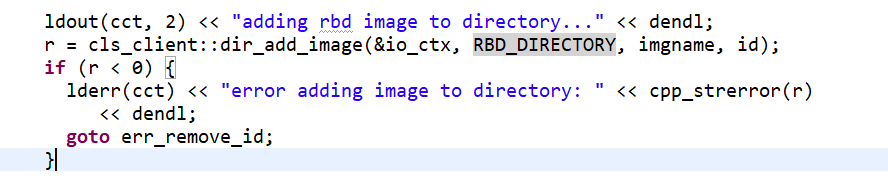
创建rbd\_id.{volume\_name}的object：



然后想这个object写入block\_name\_prefix中的id号：

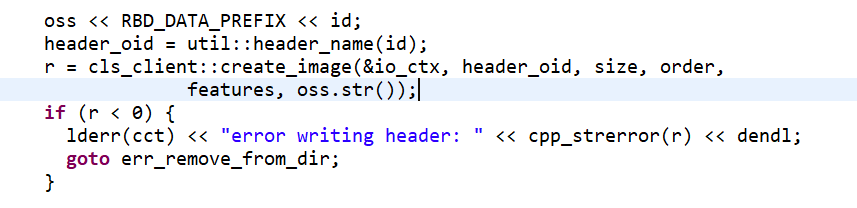


然后向rbd\_directory写入卷名和id的一一映射。

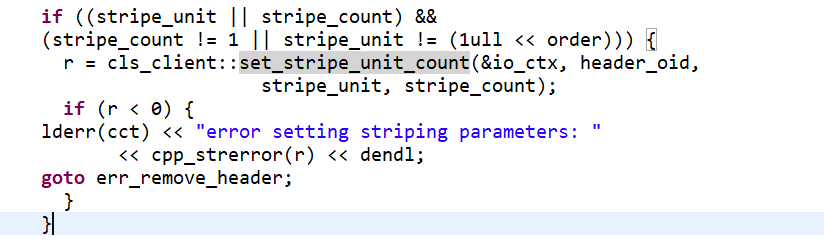


创建名为rbd\_header.id的object，并向这个object写入size,order,features,

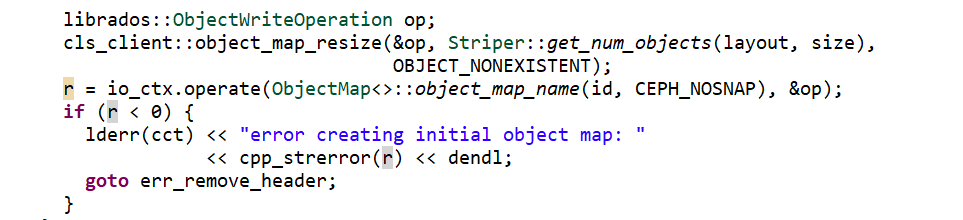
RBD\_DATA\_PREFIX等信息。



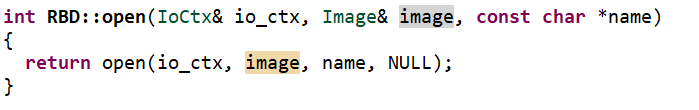
如果有条带化，则会设置条带化信息：

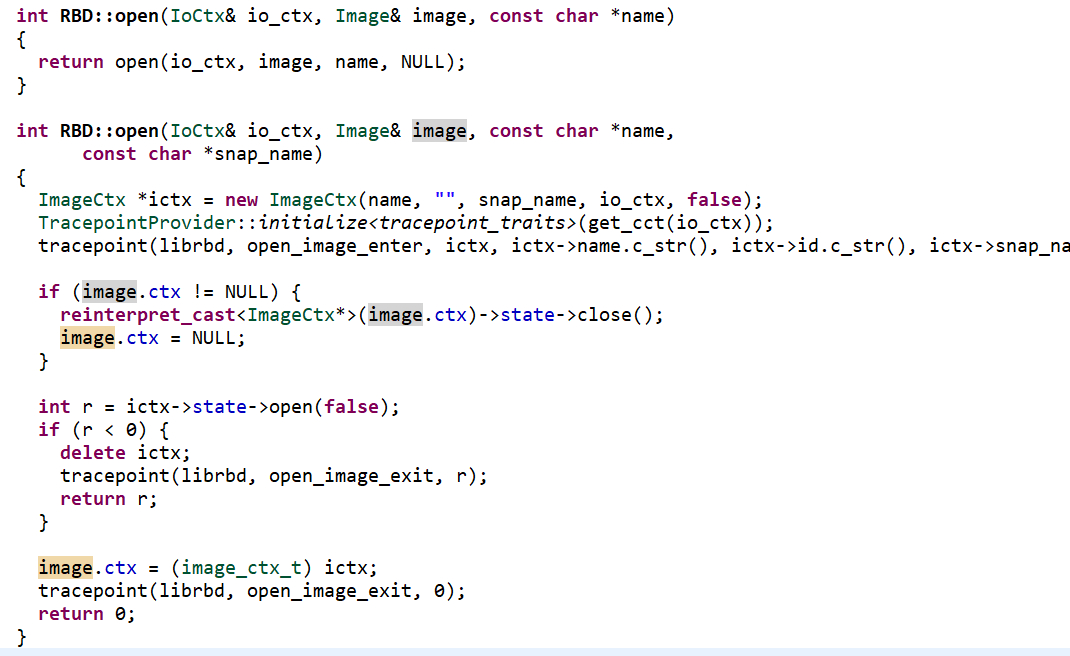


创建名为rbd\_object\_map.{id}的对象：



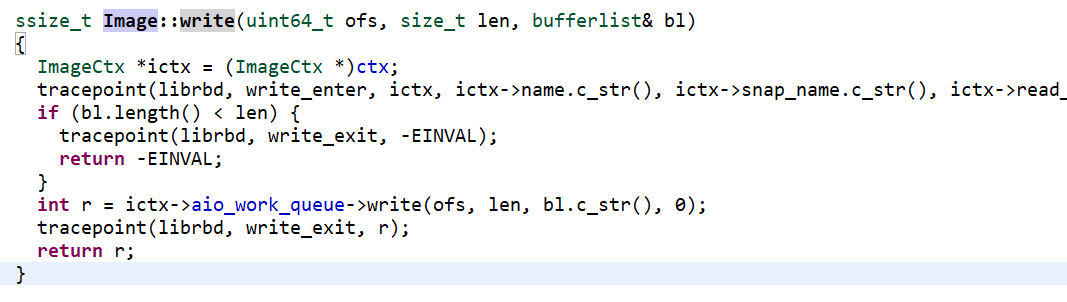
1. rbd卷的打开



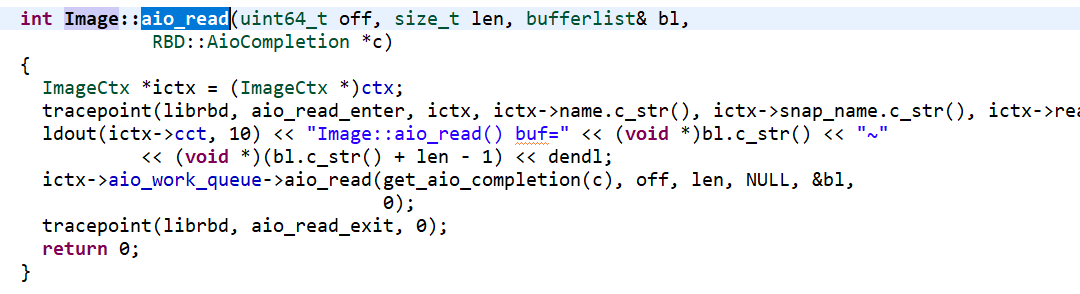


其实就是生成一个ImageCtx实例，调用其open接口。

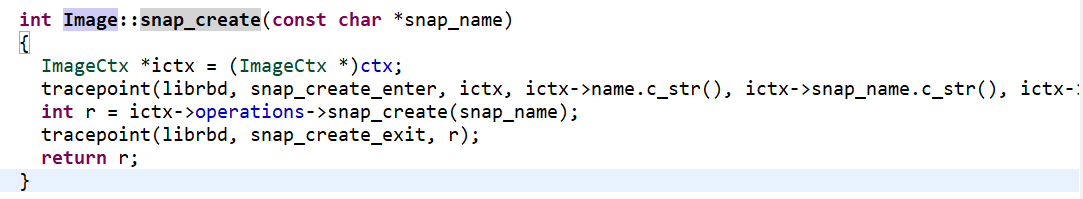
1. rbd卷的写



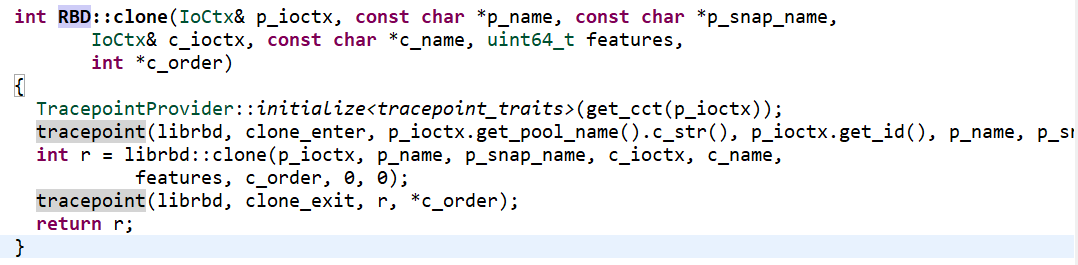
1. rbd卷的读



1. rbd卷的快照



1. rbd卷的克隆



1. rbd卷的删除

