第 1 章 Chapter 1

Ceph 整体架构

本章从比较高的层次对 Ceph 的发展历史、Ceph 的设计目标、整体架构进行简要介绍。其次介绍 Ceph 的三种对外接口：块存储、对象存储、文件存储。还介绍 Ceph 的存储基石 RADOS 系统的一些基本概念、各个模块组成和功能。最后介绍了对象的寻址过程和数据读写的原理，以及 RADOS 实现的数据服务等。

1.1 Ceph 的发展历程

Ceph 项目起源于其创始人 Sage Weil 在加州大学 Santa Cruz 分校攻读博士期间的研究课题。项目的起始时间为 2004 年，在 2006 年基于开源协议开源了 Ceph 的源代码。Sage Weil 也相应成立了 Inktank 公司专注于 Ceph 的研发。在 2014 年 5 月，该公司被 Red Hat 收购。Ceph 项目的发展历程如图 1-1 所示。

2012 年，Ceph 发布了第一个稳定版本。2014 年 10 月，Ceph 开发团队发布了 Ceph 的第七个稳定版本 Giant。到目前为止，社区平均每三个月发布一个稳定版本，目前的最新版本为 10.2.1。

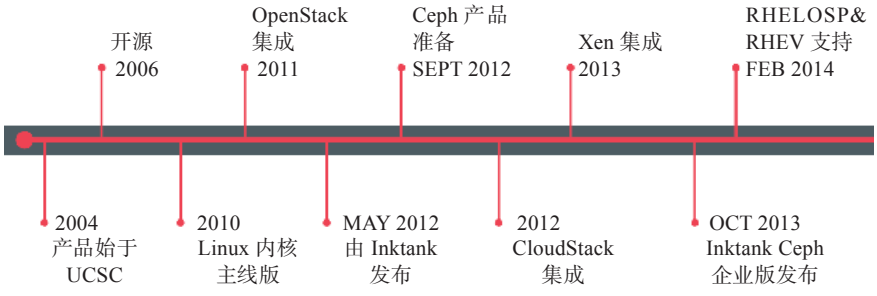

图 1-1 Ceph 的发展历程

1.2 Ceph 的设计目标

Ceph 的设计目标是采用商用硬件（Commodity Hardware）来构建大规模的、具有高可用性、高可扩展性、高性能的分布式存储系统。

商用硬件一般指标准的 x86 服务器，相对于专用硬件，性能和可靠性较差，但由于价格相对低廉，可以通过集群优势来发挥高性能，通过软件的设计解决高可用性和可扩展性。标准化的硬件可以极大地方便管理，且集群的灵活性可以应对多种应用场景。

系统的高可用性指的是系统某个部件失效后，系统依然可以提供正常服务的能力。一般用设备部件和数据的冗余来提高可用性。Ceph 通过数据多副本、纠删码来提供数据的冗余。

高可扩展性是指系统可以灵活地应对集群的伸缩。一般指两个方面，一方面指集群的容量可以伸缩，集群可以任意地添加和删除存储节点和存储设备；另一方面指系统的性能随集群的增加而线性增加。

大规模集群环境下，要求 Ceph 存储系统的规模可以扩展到成千上万个节点。当集群规模达到一定程度时，系统在数据恢复、数据迁移、节点监测等方面会产生一系列富有挑战性的问题。

1.3 Ceph 基本架构图

Ceph 的整体架构由三个层次组成：最底层也是最核心的部分是 RADOS 对象存储系统。

第二层是 librados 库层；最上层对应着 Ceph 不同形式的存储接口实现，架构如图 1-2 所示。

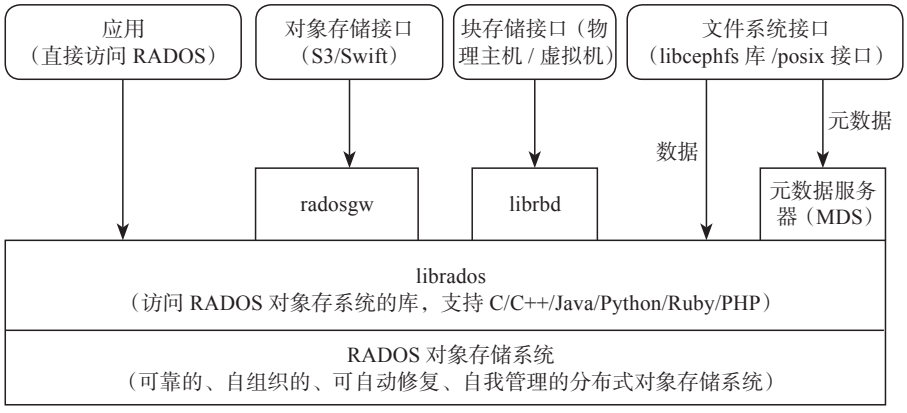

图 1-2　Ceph 基本架构图

Ceph 的整体架构大致如下：

- 最底层基于 RADOS(reliable,autonomous,distributed object store)，它是一个可靠的、自组织的、可自动修复、自我管理的分布式对象存储系统。其内部包括 ceph-osd 后台服务进程和 ceph-mon 监控进程。
- 中间层 librados 库用于本地或者远程通过网络访问 RADOS 对象存储系统。它支持多种语言，目前支持 C/C++ 语言、Java、Python、Ruby 和 PHP 语言的接口。
- 最上层面向应用提供 3 种不同的存储接口：
  - 块存储接口，通过 librbd 库提供了块存储访问接口。它可以为虚拟机提供虚拟磁盘，或者通过内核映射为物理主机提供磁盘空间。
  - 对象存储接口，目前提供了两种类型的 API，一种是和 AWS 的 S3 接口兼容的 API，另一种是和 OpenStack 的 Swift 对象接口兼容的 API。
  - 文件系统接口，目前提供两种接口，一种是标准的 posix 接口，另一种通过 libcephfs 库提供文件系统访问接口。文件系统的元数据服务器 MDS 用于提供元数据访问。数据直接通过 librados 库访问。

1.4　Ceph 客户端接口

Ceph 的设计初衷是成为一个分布式文件系统，但随着云计算的大量应用，最终变成

支持三种形式的存储：块存储、对象存储、文件系统，下面介绍它们之间的区别。

1.4.1 RBD

RBD（rados block device）是通过 librbd 库对应用提供块存储，主要面向云平台的虚拟机提供虚拟磁盘。传统 SAN 就是块存储，通过 SCSI 或者 FC 接口给应用提供一个独立的 LUN 或者卷。RBD 类似于传统的 SAN 存储，都提供数据块级别的访问。

目前 RBD 提供了两个接口，一种是直接在用户态实现，通过 QEMU Driver 供 KVM 虚拟机使用。另一种是在操作系统内核态实现了一个内核模块。通过该模块可以把块设备映射给物理主机，由物理主机直接访问。

块存储用作虚拟机的硬盘，其对 I/O 的要求和传统的物理硬盘类似。一个硬盘应该是能面向通用需求的，既能应付大文件读写，也能处理好小文件读写。也就是说，块存储既需要有较好的随机 I/O，又要求有较好的顺序 I/O，而且对延迟有比较严格的要求。

1.4.2 CephFS

CephFS 通过在 RADOS 基础之上增加了 MDS（Metadata Server）来提供文件存储。它提供了 libcephfs 库和标准的 POSIX 文件接口。CephFS 类似于传统的 NAS 存储，通过 NFS 或者 CIFS 协议提供文件系统或者文件目录服务。

Ceph 最初的设计为分布式文件系统，其通过动态子树的算法实现了多元数据服务器，但是由于实现复杂，目前还远远不能使用。目前可用于生产环境的是最新 Jewel 版本的 CephFS 为主从模式（Master-Slave）的元数据服务器。

1.4.3 RadosGW

RadosGW 基于 librados 提供了和 Amazon S3 接口以及 OpenStack Swift 接口兼容的对象存储接口。可将其简单地理解为提供基本文件（或者对象）的上传和下载的需求，它有两个特点：

- ❑ 提供 RESTful Web API 接口。
- ❑ 它采用扁平的数据组织形式。

1. RESTful 的存储接口

其接口值提供了简单的 GET、PUT、DEL 等其他接口，对应对象文件的上传、下载、删除、查询等操作。可以看出，对象存储的 I/O 接口相对比较简单，其 I/O 访问模型都是顺序 I/O 访问。

2. 扁平的数据组织形式

NAS 存储提供给应用的是一个文件系统或者是一个文件夹，其实质就是一个层级化的树状存储组织模式，其嵌套层级和规模在理论上都不受限制。

这种树状的目录结构为早期本地存储系统设计的信息组织形式，比较直观，容易理解。但是随着存储系统规模的不断扩大，特别是到了云存储时代，其难以大规模扩展的缺点就暴露了出来。相比于 NAS 存储，对象存储放弃了目录树结构，采用了扁平化组织形式（一般为三级组织结构），这有利于实现近乎无限的容量扩展。它使用业界标准互联网协议，更加符合面向云服务的存储、归档和备份需求。

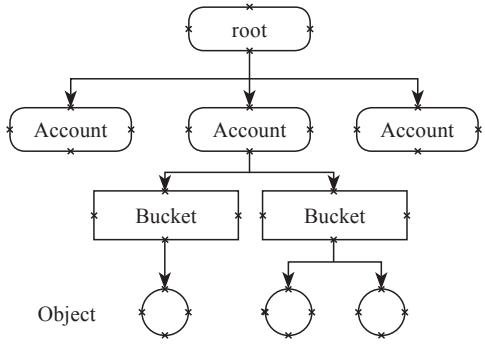

图 1-3　Amazon S3 的对象存储结构

由于 Amazon 在云存储领域的影响力，Amazon 的 S3 接口已经成为事实上的对象存储的标准接口。如图 1-3 所示，其接口分三级存储：Account/Bucket/Object（账户 / 桶 / 对象）。一个 Account 可以看作一个用户（租户），其下可以包含若干个的 Bucket，一个 Bucket 可以拥有若干对象，其数量在理论上都不受限制。

在云计算领域，OpenStack 已经成为广泛采用的云计算管理系统，OpenStack 的对象存储接口 Swift 也成为广泛采用的接口，如图 1-4 所示，其也采用分三级存储：Account/

Container/Object（账户 / 容器 / 对象），每层节点数均没有限制。可以看出，Swift 接口和 S3 类似，Swift 的 Container 对应 S3 的 Bucket 概念。Swift 接口和 S3 接口没有太大的区别，但是一些管理接口会有一些差别。

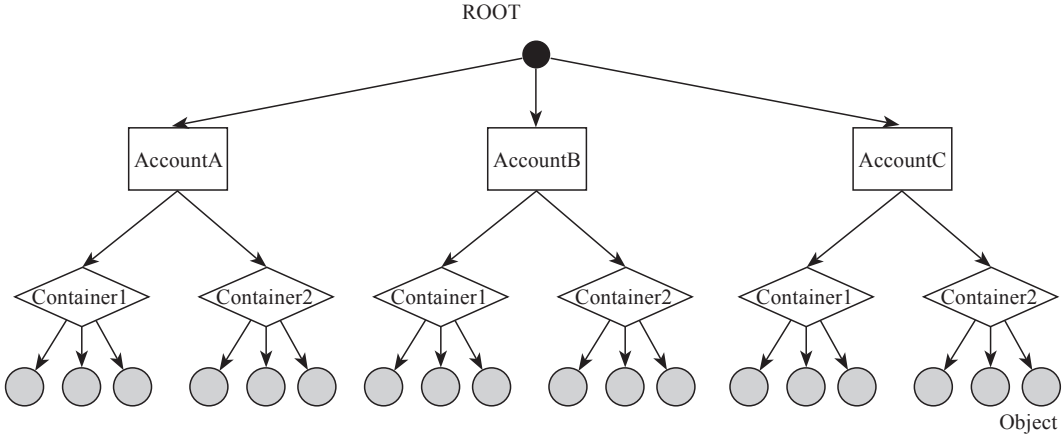

图 1-4　OpenStack Swift 对象存储结构

1.5　RADOS

RADOS 是 Ceph 存储系统的基石，是一个可扩展的、稳定的、自我管理的、自我修复的对象存储系统，是 Ceph 存储系统的核心。它完成了一个存储系统的核心功能，包括：Monitor 模块为整个存储集群提供全局的配置和系统信息；通过 CRUSH 算法实现对象的寻址过程；完成对象的读写以及其他数据功能；提供了数据均衡功能；通过 Peering 过程完成一个 PG 内存达成数据一致性的过程；提供数据自动恢复的功能；提供克隆和快照功能；实现了对象分层存储的功能；实现了数据一致性检查工具 Scrub。下面分别对上述基本功能做简要的介绍。

1.5.1　Monitor

Monitor 是一个独立部署的 daemon 进程。通过组成 Monitor 集群来保证自己的高可用。Monitor 集群通过 Paxos 算法实现了自己数据的一致性。它提供了整个存储系统的节点信息等全局的配置信息。

Cluster Map 保存了系统的全局信息，主要包括：

- Monitor Map
  - 包括集群的 fsid
  - 所有 Monitor 的地址和端口
  - current epoch
- OSD Map：所有 OSD 的列表，和 OSD 的状态等。
- MDS Map：所有的 MDS 的列表和状态。

1.5.2　对象存储

这里所说的对象是指 RADOS 对象，要和 RadosGW 的 S3 或者 Swift 接口的对象存储区分开来。对象是数据存储的基本单元，一般默认 4MB 大小。图 1-5 就是一个对象的示意图。

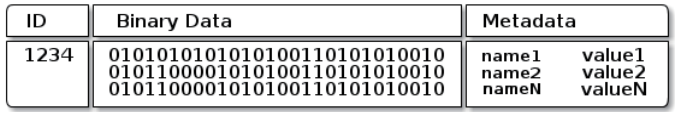

图 1-5　对象示意图

一个对象由三个部分组成：

- 对象标志（ID），唯一标识一个对象。
- 对象的数据，其在本地文件系统中对应一个文件，对象的数据就保存在文件中。
- 对象的元数据，以 Key-Value（键值对）的形式，可以保存在文件对应的扩展属性中。由于本地文件系统的扩展属性能保存的数据量有限制，RADOS 增加了另一种方式：以 Leveldb 等的本地 KV 存储系统来保存对象的元数据。

1.5.3　pool 和 PG 的概念

pool 是一个抽象的存储池。它规定了数据冗余的类型以及对应的副本分布策略。目前实现了两种 pool 类型：replicated 类型和 Erasure Code 类型。一个 pool 由多个 PG 构成。

PG（placement group）从名字可理解为一个放置策略组，它是对象的集合，该集合里

的所有对象都具有相同的放置策略：对象的副本都分布在相同的 OSD 列表上。一个对象只能属于一个 PG，一个 PG 对应于放置在其上的 OSD 列表。一个 OSD 上可以分布多个 PG。

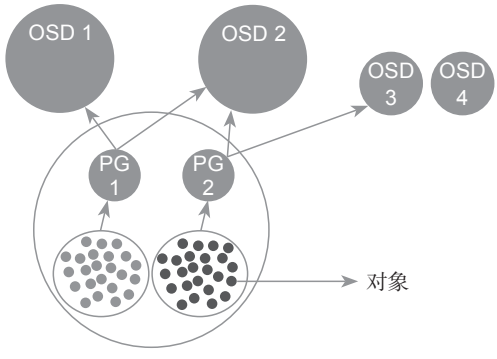

图 1-6　PG 的概念示意图

PG 的概念如图 1-6 所示，其中：

- PG1 和 PG2 都属于同一个 pool，所以都是副本类型，并且都是两副本。
- PG1 和 PG2 里都包含许多对象，PG1 上的所有对象，主从副本分布在 OSD1 和 OSD2 上，PG2 上的所有对象的主从副本分布在 OSD2 和 OSD3 上。
- 一个对象只能属于一个 PG，一个 PG 包含多个对象。
- 一个 PG 的副本分布在对应的 OSD 列表中。在一个 OSD 上可以分布多个 PG。示例中 PG1 和 PG2 的从副本都分布在 OSD2 上。

1.5.4　对象寻址过程

对象寻址过程指的是查找对象在集群中分布的位置信息，过程分为两步：

1）对象到 PG 的映射。这个过程是静态 hash 映射（加入 pg split 后实际变成了动态 hash 映射方式），通过对 object_id，计算出 hash 值，用该 pool 的 PG 的总数量 pg_num 对 hash 值取模，就可以获得该对象所在的 PG 的 id 号：

```
pg_id =  hash( object_id )  %  pg_num
```

2）PG 到 OSD 列表映射。这是指 PG 上对象的副本如何分布在 OSD 上。它使用 Ceph 自己创新的 CRUSH 算法来实现，本质上是一个伪随机分布算法。

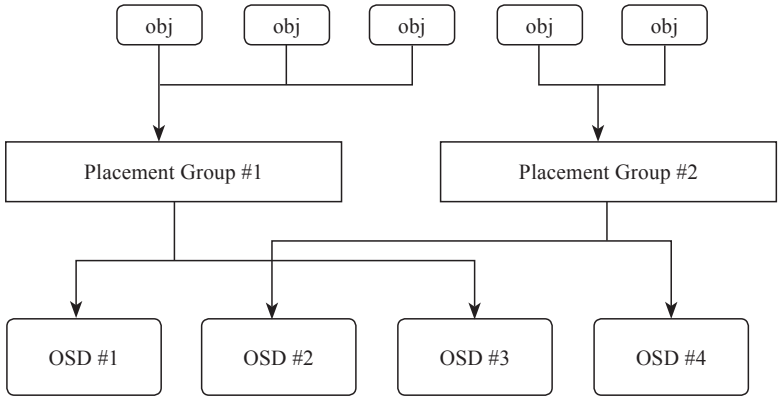

图 1-7　对象寻址过程

如图 1-7 所示的对象寻址过程：

1）通过 hash 取模后计算，前三个对象分布在 PG1 上，后两个对象分布在 PG2 上。

2）PG1 通过 CRUSH 算法，计算出 PG1 分布在 OSD1、OSD3 上；PG2 通过 CRUSH 算法分布在 OSD2 和 OSD4 上。

1.5.5　数据读写过程

Ceph 的数据写操作如图 1-8 所示，其过程如下：

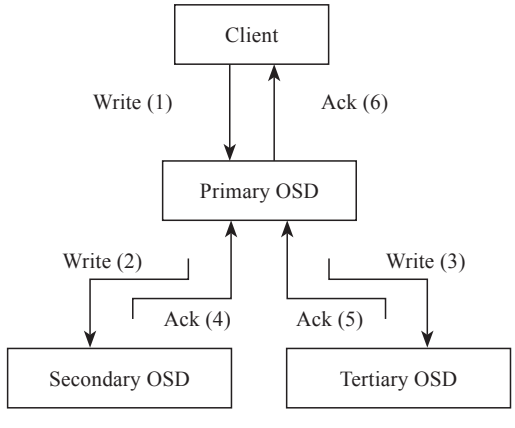

图 1-8　读写过程

1）Client 向该 PG 所在的主 OSD 发送写请求。

2）主 OSD 接收到写请求后，同时向两个从 OSD 发送写副本的请求，并同时写入主 OSD 的本地存储中。

3）主 OSD 接收到两个从 OSD 发送写成功的 ACK 应答，同时确认自己写成功，就向客户端返回写成功的 ACK 应答。

在写操作的过程中，主 OSD 必须等待所有的从 OSD 返回正确应答，才能向客户端返回写操作成功的应答。由于读操作比较简单，这里就不介绍了。

1.5.6 数据均衡

当在集群中新添加一个 OSD 存储设备时，整个集群会发生数据的迁移，使得数据分布达到均衡。Ceph 数据迁移的基本单位是 PG，即数据迁移是将 PG 中的所有对象作为一个整体来迁移。

迁移触发的流程为：当新加入一个 OSD 时，会改变系统的 CRUSH Map，从而引起对象寻址过程中的第二步，PG 到 OSD 列表的映射发生了变化，从而引发数据的迁移。

举例来说明数据迁移过程，表 1-1 是数据迁移前的 PG 分布，表 1-2 是数据迁移后的 PG 分布。

表 1-1 数据迁移前的 PG 分布

	OSD1	OSD2	OSD3
PGa	PGa1	PGa2	PGa3
PGb	PGb3	PGb1	PGb2
PGc	PGc2	PGc3	PGc1
PGd	PGd1	PGd2	PGd3

表 1-2 数据迁移后的 PG 分布

	OSD1	OSD2	OSD3	OSD4
PGa	~~PGa1~~	PGa2	PGa3	PGa1
PGb	PGb3	PGb1	~~PGb2~~	PGb2
PGc	PGc2	~~PGc3~~	PGc1	PGc3
PGd	PGd1	PGd2	PGd3	

当前系统有 3 个 OSD，分布为 OSD1、OSD2、OSD3；系统有 4 个 PG，分布为 PGa、

PGb、PGc、PGd；PG 设置为三副本：PGa1、PGa2、PGa3 分别为 PGa 的三个副本。PG 的所有分布如表 1-1 所示，每一行代表 PG 分布的 OSD 列表。

当添加一个新的 OSD4 时，CRUSH Map 变化导致通过 CRUSH 算法来计算 PG 到 OSD 的分布发生了变化。如表 1-2 所示：PGa 的映射到了列表 [OSD4，OSD2，OSD3] 上，导致 PGa1 从 OSD1 上迁移到了 OSD4 上。同理，PGb2 从 OSD3 上迁移到 OSD4，PGc3 从 OSD2 上迁移到 OSD4 上，最终数据达到了基本平衡。

1.5.7 Peering

当 OSD 启动，或者某个 OSD 失效时，该 OSD 上的主 PG 会发起一个 Peering 的过程。Ceph 的 Peering 过程是指一个 PG 内的所有副本通过 PG 日志来达成数据一致的过程。当 Peering 完成之后，该 PG 就可以对外提供读写服务了。此时 PG 的某些对象可能处于数据不一致的状态，其被标记出来，需要恢复。在写操作的过程中，遇到处于不一致的数据对象需要恢复的话，则需要等待，系统优先恢复该对象后，才能继续完成写操作。

1.5.8 Recovery 和 Backfill

Ceph 的 Recovery 过程是根据在 Peering 的过程中产生的、依据 PG 日志推算出的不一致对象列表来修复其他副本上的数据。

Recovery 过程的依据是根据 PG 日志来推测出不一致的对象加以修复。当某个 OSD 长时间失效后重新加入集群，它已经无法根据 PG 日志来修复，就需要执行 Backfill（回填）过程。Backfill 过程是通过逐一对比两个 PG 的对象列表来修复。当新加入一个 OSD 产生了数据迁移，也需要通过 Backfill 过程来完成。

1.5.9 纠删码

纠删码（Erasure Code）的概念早在 20 世纪 60 年代就提出来了，最近几年被广泛应用在存储领域。它的原理比较简单：将写入的数据分成 N 份原始数据块，通过这 N 份原始数据块计算出 M 份效验数据块，N+M 份数据块可以分别保存在不同的设备或者节点中。可以允许最多 M 个数据块失效，通过 N+M 份中的任意 N 份数据，就还原出其他数据块。

目前 Ceph 对纠删码（EC）的支持还比较有限。RBD 目前不能直接支持纠删码（EC）模式。其或者应用在对象存储 radosgw 中，或者作为 Cache Tier 的二层存储。其中的原因和具体实现都将在后面的章节详细介绍。

1.5.10 快照和克隆

快照（snapshot）就是一个存储设备在某一时刻的全部只读镜像。克隆（clone）是在某一时刻的全部可写镜像。快照和克隆的区别在于快照只能读，而克隆可写。

RADOS 对象存储系统本身支持 Copy-on-Write 方式的快照机制。基于这个机制，Ceph 可以实现两种类型的快照，一种是 pool 级别的快照，给整个 pool 中的所有对象统一做快照操作。另一种就是用户自己定义的快照实现，这需要客户端配合实现一些快照机制。RBD 的快照实现就属于后者。

RBD 的克隆实现是在基于 RBD 的快照基础上，在客户端 librbd 上实现了 Copy-on-Write（cow）克隆机制。

1.5.11 Cache Tier

RADOS 实现了以 pool 为基础的自动分层存储机制。它在第一层可以设置 cache pool，其为高速存储设备（例如 SSD 设备）。第二层为 data pool，使用大容量低速存储设备（如 HDD 设备）可以使用 EC 模式来降低存储空间。通过 Cache Tier，可以提高关键数据或者热点数据的性能，同时降低存储开销。

Cache Tier 的结构如图 1-9 所示，说明如下：

- ❑ Ceph Client 对于 Cache 层是透明的。
- ❑ 类 Objecter 负责请求是发给 Cache Tier 层，还是发给 Storage Tier 层。
- ❑ Cache Tier 层为高速 I/O 层，保存热点数据，或称为活跃的数据。
- ❑ Storage Tier 层为慢速层，保存非活跃的数据。
- ❑ 在 Cache Tier 层和 Storage Tier 层之间，数据根据活跃度自动地迁移。

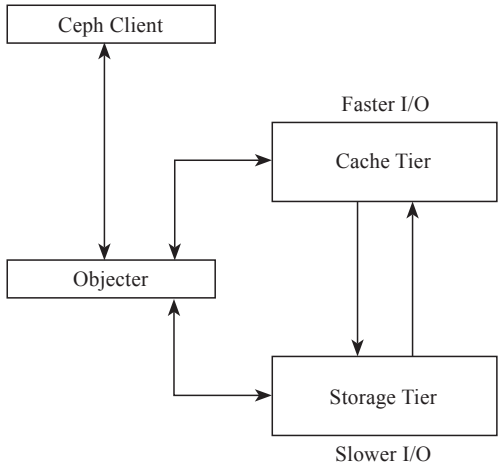

图 1-9　Cache Tier 结构图

1.5.12　Scrub

Scrub 机制用于系统检查数据的一致性。它通过在后台定期（默认每天一次）扫描，比较一个 PG 内的对象分别在其他 OSD 上的各个副本的元数据和数据来检查是否一致。根据扫描的内容分为两种，第一种是只比较对象各个副本的元数据，它代价比较小，扫描比较高效，对系统影响比较小。另一种扫描称为 deep scrub，它需要进一步比较副本的数据内容检查数据是否一致。

1.6　本章小结

本章介绍了 Ceph 的系统架构，通过本章，可以对 Ceph 的基本架构和各个模块的组件有了整体的了解，并对一些基本概念及读写的原理、各个数据功能模块有了大致了解。

本章介绍的 Ceph 客户端的基本概念，在第 5 章会详细介绍到。本章介绍的对象寻址的 CRUSH 算法将会在第 4 章详细介绍到。在本章介绍的本地对象存储将会在第 7 章详细介绍到。本章介绍的数据读写流程，将会在第 6 章详细介绍。本章介绍的纠删码将在第 8 章中详细介绍。本章介绍的快照和克隆将在第 9 章详细介绍。本章介绍的 Ceph Peering 的过程将会在第 10 章详细介绍。本章介绍的数据恢复和回填将会在第 11 章介绍到。本章介绍的 Ceph Srcub 机制将会在第 12 章详细介绍。本章介绍的 Cache Tier 将在第 13 章详细介绍。