

# 中国科学技术大学

# 工程硕士学位论文



## 基于 ceph 的云存储运维系统的设计 和实现

作者姓名：	孙君伟
学科专业：	软件工程
校内导师：	周英华 高级工程师
企业导师：	胡国华 高级工程师
完成时间：	二〇一六年八月十七日

University of Science and Technology of China  
A dissertation for master's degree  
of engineering



**Design and implementation of  
cloud storage operation and  
maintenace system based on  
ceph**

Author's Name: Junwei Sun

Speciality: Software Engineering

Supervisor: Yinghua Zhou Senior Engineer

Advisor: Guohua Hu Senior Engineer

Finished time: Aug 17<sup>nd</sup>, 2016

## 中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外,论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名: \_\_\_\_\_

签字日期: \_\_\_\_\_

## 中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一,学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权,即:学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版,允许论文被查阅和借阅,可以将学位论文编入有关数据库进行检索,可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

☐公开    ☐保密 (\_\_\_\_年)

作者签名: \_\_\_\_\_

签字日期: \_\_\_\_\_

## 摘 要

此次主要讨论基于 ceph 的云存储运维系统的设计和实现。基于 ceph 的云存储运维系统，是为了实现将网络中相互独立，型号各异，价格低廉的存储设备统一组织起来，并对外提供统一接口。从而形成具有备份、容灾、易于维护特性的存储系统。下面叙述下系统的主要特性。

在 ceph 层提供的主要特性有（1）无单点瓶颈：基于 ceph 的存储机制，在数据存取时，只需要结合已有信息和需求，通过对应的算法，自己就可以知道数据对应的存取位置；（2）高效存取：对于大文件，如何有效存储至关重要，这不仅仅关系到数据存储的安全，还包括读取数据的效率，基于 ceph 的云存储运维系统，通过切片，实现均匀、高效的存取；（3）负载均衡：在数据存储的过程中，难免会出现磁盘损坏，磁盘扩容，以及不同磁盘使用情况的失衡，这样将会影响到数据存取效率，如果能做到将同一个大文件，合理的存储在不同磁盘上，将会使数据存储效率有明显的提升，充分利用系统提供的高并发特性，实现数据的并发存取。

在用户界面层将实现和用户交互的监控界面，隐藏底层操作细节，通过用户的点击实现对应的操作。

**关键字：**容灾备份，均衡，并发

## Abstraction

The design and implementation of cloud storage operation and maintenance system based on CEPH is mainly discussed. In order to achieve the network in each other independent, different models, inexpensive storage devices unified organization, and to provide a unified interface to the outside world, in order to achieve an easy storage, backup and disaster tolerance system maintenance characteristics, the cloud storage system based on CEPH was appearance. The main characteristics of the system are described below.

The main features provided in the CEPH layer are that (1) no single point bottleneck: CEPH based storage mechanism, in the data access, only need to combine the existing information and needs, through the corresponding algorithm, you can know the data corresponding to the access location; (2) efficient access: For big files, effectively store is essential, It is not only related to the safety of data storage, but also includes the efficiency of reading data, CEPH based cloud storage operation and maintenance system, through the slice, to achieve uniform, efficient access; (3) load balance: In the process of data storage, it is inevitable that there will be a disk damage, disk expansion, as well as the disk of different disk use rate of imbalance, which will affect the efficiency of data access, If you can store a big file in a good method, the data storage efficiency will be significantly improved. These will full use of the system to realize the character of the high concurrency, the realization of data access.

In the user interface layer, I will be implemented the function that used monitor the state of disk, hide the underlying operating details, through the user's click to achieve the corresponding operation.

**Key Words:** Backup, balanced, concurren

# 目录

第一章 绪论.....	1
1.1 选题的依据与意义.....	1
1.2 国内外研究发展状况及发展趋势.....	2
1.3 云存储系统设计的先进性和意义.....	4
1.4 本人主要工作.....	5
1.5 论文组织.....	5
第二章 理论概念.....	6
2.1 云存储.....	6
2.1.1 云存储出现场景.....	6
2.1.2 云存储的关键技术.....	6
2.2 ceph 机制.....	7
2.2.1 集群框架.....	7
2.2.2 存储机制.....	8
2.3 总结.....	11
第三章 需求分析.....	12
3.1 系统概述.....	12
3.1.1 前端概述.....	12
3.1.2 后端概述.....	14
3.1.3 系统构造.....	14
3.2 系统总体架构.....	15
3.3 云存储运维系统性能需求分析.....	16
3.3.1 高性能需求.....	17
3.3.2 高可用性需求.....	17
3.3.3 高扩展性需求.....	18

3.4 云存储运维系统功能需求分析.....	18
3.5 总结.....	19
第四章 系统设计.....	20
4.1 系统架构设计.....	20
4.1.1 前端架构设计.....	20
4.1.2 后端架构设计.....	21
4.2 云存储底层设计.....	21
4.3 总结.....	23
第五章 系统的部署及管理.....	24
5.1 云存储环境搭建和部署.....	24
5.1.1 ceph 存储系统的搭建和部署.....	24
5.1.2 calamari 框架的搭建.....	38
5.1.3 Java 框架的搭建.....	39
5.1.4 前端框架的搭建.....	40
5.2 系统代码的管理.....	40
5.3 总结.....	42
第六章 系统开发和实现.....	43
6.1 开发工具及环境.....	43
6.2 云存储运维系统的开发与实现.....	43
6.2.1 前期准备.....	43
6.2.2 运维系统的实现.....	48
6.3 总结.....	56
第七章 总结.....	57
参考文献.....	63
附 录.....	65
致 谢.....	67

## 第一章 绪论

### 1.1 选题的依据与意义

个人需求:当今社会,互联网几乎已在各个国家都得到了普及,数据信息,比如照片、电影、工作文件等数据资料自然而然的出现,这就需要存储设备去保存这些信息,且随着信息社会快速的发展,人们需求不断扩大,对他们的要求也不断提高,这些数据信息的体积也越来越大。U 盘、光盘、硬盘等凭借大容量,相对体积小,价格适中,存储稳定等特性轻易的战胜了传统的软盘,成为当代青年日常生活必需品。而随着网络磁盘的到来,很多人又将目光转移到了网盘。它相对于当代实体存储介质,有不可比拟的优点。当采用网盘存储数据时,无论在哪,只要能联网,就随时可用联网终端设备,存取自己的资料。也为多人协同工作带来了极大的便利。很多网盘,比如 360 云盘、百度网盘等都提供免费的,较大容量的网盘,对于普通用户,完全可以满足日常需求。这是云存储给个人带来的便利。云存储使人们找到了一种和物理介质存储设备一样,具有存储性能的方式。同时,云存储也避免出现因物理设备的损坏或丢失而造成的不可恢复的灾难。当代网络的普及使得云存储变得异常方便,通过手机终端就可以说实现数据的存取,相当于手机拥有了无限大的存储空间。极大方便用户对存储空间的需求。

大型应用:随着云计算和物联网的兴起,智能应用成为当今一大主题。智能应用主要涉及到信息的收集,整理和应用。因涉及范围广,需求各异,会形成海量、复杂的信息数据。这些信息如果还是利用传统模式,将信息存储在由政府,企业提供的孤立的存储介质上,那这种代价将会非常高<sup>[1]</sup>。随着后期数据量和数据种类的不断增加,对磁盘管理起来会越来越复杂。在采购存储介质方面,前期面临着存储介质被闲置的浪费,后期因为设备紧张,还要有专门人员负责购买昂贵的储存介质。使得政府、企业前期投入特别大。同时,配备有专门的人员去维护。存储介质的升级换代也是个很大的问题。信息的可靠性也无法得到保证。综合各种因素,采用普通的,孤立的存储介质将会花费高昂的代价,效果也不理想。这些弊端通过云存储系统就可以很好的解决。云存储系统



可以按需分配磁盘空间，后期可以轻松扩容，在数据安全上也会有很好的保证。在管理上有专门的人员负责。这样就不会存在因一次性购买大量存储介质而造成的浪费，也不用为扩容时去买新的存储介质。花费代价上远小于自己独立管理。给政府，企业带来极大便利。下面通过生活中一个例子去感受下。

当代数字化迅速发展的今天，安全防护监控在日常生活中的需求也在不断升级。安全监控将会产生大量数据<sup>[2]</sup>。随着社会的发展，与安全监控匹配的存储也经历好几代的发展，从初始的 VCR 模拟存储到 DVR 数字数据存储，又从 DVR 数字数据存储到现在的网络集中存储。因分辨率的提高，现在产生的数据已达到 PB 级别。对于这么大的数据，采用传统的存储模式，很难做到对这些数据高效的存取。如果采用基于 ceph 的分布式的云存储运维系统，可以将这些数据均匀的分布存储在各个存储节点上，当对数据进行存取时，可以实现所有含有对应信息的节点，并发、高效的进行数据存取操作。使得政府、企业在这方面的需求得到了很好的解决。

### 1.2 国内外研究发展状况及发展趋势

ceph 概述:此次毕业论文设计,是基于 ceph<sup>[3]</sup>分布式系统的,接下来看下 ceph 分布式系统在国内外现状和发展。ceph 是一种具有高性能,高扩展性和高可用性的分布式存储系统。可以提供对象存储,块存储和文件系统存储三种存储模式<sup>[4]</sup>。从而使其满足不同应用对应的存储需求。ceph 可以部署在上千台服务器上。ceph 项目由 Sage Weil 在加州大学 Santa Cruz 分校攻读博士期间研究的课题。该项目是从 2004 年开始,在 2006 年的 OSDI 学术会议上, Sage Weil 发表了介绍 ceph 的论文,并提供了 ceph 项目的下载链接。从此,ceph 开始广为人知。ceph 分布式系统是由 Sage Weil 在 2007 年提出,作为其博士论文设计的新一代分布式系统。在 2010 年三月,应用到 Linux 内核。随着 ceph 的不断发展, Sage Weil 于 2011 年创建了 Inktank 公司,主导 ceph 的开发和社区维护。随后,ceph 分布式系统的发布周期为三个月。ceph 分布式系统虽然非常优秀,但随着 ceph 的不断发展,还是有些问题需要改进或优化。比如:(1) 数据双倍写入的性能:因为 ceph 分布式系统为了支持事务,引入了日志机制。也就是说,一份数据需要写两遍,即日志和本地系统文件,使得实际的磁盘输出的吞吐量只有物理性能的一半。(2) IO 路径太长:这种情况在客户端和服务端都有存在,比如对

于 osd, 一个 IO 的操作需要经过 message、OSD、FileJournal、FileStore 多个部分才可以实现整个过程, 而每个部分都会涉及到队列和线程的切换, 有些模块在对 IO 进行处理时还会涉及内存拷贝, 因此造成整体性能不高。(3) 兼容性: ceph 分布式系统刚开始是基于 HDD 设计的, 所以没有全面考虑到 SSD。所以没有能充分发挥新型存储介质的性能。特别是在延迟和 IOPS 方面, 问题较为突出。ceph 分布式操作系统对外提供丰富的操作接口, 给管理网络磁盘提供了便利。使得基于 ceph 的云存储运维系统自然而然的出现。

基于 ceph 的云存储运维系统的现状: 基于 ceph 的云存储运维系统, 是在 ceph 分布式系统的基础上, 通过 calamari, 做了进一步封装, 实现了对磁盘信息的监控和部分操作。不同厂商会提供不同用途的存储设备, 根据不同用途, 会有不同的监控系统, 其主要重点实现是在云存储系统上。下面来了解下基于 ceph 的云存储系统的一些现状。它在 ceph 分布式系统和云计算的基础上发展而来的, 通过集群的应用, 将网络上不同的存储介质统一组织起来, 并对外提供存储功能的一个新兴存储方式。相对于传统的孤立存储, 基于 ceph 的云存储系统能提供更加便携、安全、可靠、高效的性能<sup>[5]</sup>。下面将通过基于 ceph 的云存储系统的优势和劣势来分析下基于 ceph 的云存储系统的现状: (1) 优势: 基于 ceph 的云存储系统可以按客户需求, 分配对应存储空间, 且随着后期的需求的变化, 可以及时方便的进行扩容或缩容。且在对设备进行缩容或扩容时, 给客户带来良好的体验, 全程操作基本不会影响客户正常业务, 节省了客户维护方面的开销<sup>[6]</sup>。同时也节省升级服务器的费用。在数据安全方面, 云存储会自动检测故障磁盘, 当出现故障磁盘时, 系统会根据预先设计的规则, 将故障磁盘上的数据自动迁移到正常运行设备上<sup>[7]</sup>。从而保证了数据的安全, 并且不影响客户的正常操作。同时也使得故障磁盘对客户的损失降到了最低。采用云存储系统, 还享有专业的运维团队, 负责维护存储系统的稳定和安全。随着供应商的日益增多, 使用费用将会不断降低。(2) 劣势: 基于 ceph 的云存储运维系统, 在存储介质稳定性方面有很好的保证, 但在信息安全(主要指被非法用户获取)方面存在明显不足。因为数据存储是通过网络进行存储, 所以有可能被中途拦截, 同时对网络磁盘进行管理维护的工作人员可以轻易获得存储在云磁盘上的数据。虽然通过加密可以降低信息泄露的风险, 但在数据处理方面将变得比较复杂, 使得数据处理能力下降<sup>[8]</sup>。能耗大也是云存储面临的另一重大挑战。一方面, 随着基础规模的扩大, 能量消耗将会急剧增加。另一方面, 随着数据量

的增加,启动设备组件增加,所需能源越来越大。有统计,在2010年,其对电能的消耗为 $2 \times 10^{11}$ 千瓦时,约占全球总电量的1.3%,且其比例还在不断增加<sup>[9]</sup>。存储系统会在能源消耗中占很大比例,仅次于计算资源耗电量,约占30%<sup>[10]</sup>。因此,现在云存储的能耗也是云存储的一个重要的关注点。基于ceph的云存储运维系统会涉及到负载均衡,在小规模集群的环境下,在执行负载均衡操作时,有时会出现达不到预先设定的均衡<sup>[11]</sup>。

基于ceph的云存储运维系统发展趋势:(1)安全性:基于ceph的云存储运维系统在其存储方面,安全性一直以来都是基于ceph的云存储系统的最大焦点之一<sup>[12]</sup>。如何保证用户的数据不被盗用,保证用户的数据在传输过程中不被篡改,将会成为云存储发展的一个重要目标之一。只有很好的解决了这些基本的安全问题,才能让更多的用户放心选择使用基于ceph的云存储系统之上的各种应用。(2)多样性:由于客户需求的多样性,基于ceph的云存储运维系统将显示各种各样的应用,同时会使得数据存储更加复杂,这就要求基于ceph的云存储运维系统在应对将来更多需求,能提供更加丰富和完善的机制。备份、容错、恢复和快照等方面做的更加完善<sup>[13]</sup>。(3)易用性:物联网在不断发展,将会出现用特定设备访问特定数据。这将会使得基于ceph的云存储运维系统在和各个终端设备上有更好的兼容,完善用户体验。同时,采用更先进的技术提高数据读写速度。

### 1.3 云存储系统设计的先进性和意义

云存储系统设计的思想是将大文件,分成易于存取的、相同大小的块,较均匀的分布在各个存储节点上,充分利用每个存储节点的数据处理能力,实现高效、并发的对数据进行相应的处理。同时,通过内部实现机制,避免出现单点故障。每个客户节点,通过自身计算,就可以算出自己所要读取或写入文件的对应位置。彻底解决了数据存储过程中,众多请求对单个或某几个服务器造成的压力。

通过ceph云储存系统的存储机制,可以很好的应对大数据的存取操作,给用户带来良好的体验。在稳定性方面,基于ceph的云存储,拥有三份相应数据,可以根据配置信息,执行相应的存取策略,实现较稳定、安全的存储数据<sup>[14]</sup>。云存储的内部实现机制,对出现磁盘损坏有很好的应对策略<sup>[15]</sup>。在后期的扩容

过程中，几乎不被用户感知，就可以实现相应存储空间的升级<sup>[16]</sup>。

## 1.4 本人主要工作

参与了界面信息显示设计，并负责通过 Axure RP 7.0 画图软件，将整体页面结构和逻辑模拟出来。在中期的实现阶段，负责 Server SAN 界面的开发，以及对应前端和后端的交互的实现。在后期阶段，负责本模块的功能维护，还做了云存储磁盘的性能测试。

## 1.5 论文组织

在论文第二章讲述项目中所涉及到的基本概念；第三章讲述了系统整体架构和功能需求；第四章讲述了系统整体设计；第五章讲述了在实际开发者的部署以及为了展示具体细节，在自己机器上搭建的开发环境；第六章讲述了项目中开发的功能；第七章为总结，讲述了在项目开发中遇到的问题以及解决方法。以上是文章结构的主要部分。

## 第二章 理论概念

### 2.1 云存储

#### 2.1.1 云存储出现场景

随着互联网的发展，以及云计算和物联网的出现，人类产生的数据信息在以极快的速度产生。具体有多快，我们通过一个名人提出的定律感受下。图灵获奖者 Jim Gray 曾提出的数据增长经验定律——网络环境下每 18 个月产生的数据量等于有史以来数据量之和。通过他提出的定律，我们可以更好的感受下，当前社会中数据增长的速度。随之而来的问题就是如何能安全、有效的存储这些数据，同时保证在人们需要这些数据时，有能安全、高效的呈现给用户。基于 ceph 的云存储，采用分布式存储系统，无单一节点故障瓶颈，可以很好满足上面出现的需求。

#### 2.1.2 云存储的关键技术

随着不同种类数据的急剧增加和人们对信息安全需求的不断提高，拥有安全，性价比高的存储机制成为企业，政府等机构的关注的焦点。而基于 ceph 的云存储系统具备一下特性。

（1）易扩充：云储存具有按需定制所需空间。随着社会的发展，数据几乎成爆炸式的方式出现，作为公司，企业，政府他们预测不到自己将来需要多少存储。如果还是采取传统存储方式，自购存储设备，必然会出现前期存储闲置，后期存储设备紧张。这两种情况都会造成极大的浪费。而如果采用当前所提供的云储存技术，按需申请存储空间，后期出现大量数据时，可以随时申请扩容，方便、快捷<sup>[17]</sup>。避免了闲置浪费和后期因存储紧张造成的不便。同时也缓解刚刚起步公司、企业，在购置存储设备上带来的压力。

（2）易管理：随着信息量的急剧增加，存储介质也在发生翻天覆地的变化，存储之间的差异在所难免，云存储会将这些存储设备协调起来，通过云存储上

的软件优化存储性能。同时云存储提供专业化管理策略，对资源进行了整合，对外提供统一操作接口。从而极大的提高了存取效率。

(3) 高性能：云存储采用的是分布式存储方式，可以根据数据访问，使用频率，磁盘使用率的不同进行自动均衡，充分利用每个服务节点的计算资源，极大的发挥系统的并发性能<sup>[18]</sup>。极大的提高了用户体验，让客户不在为大文件的存储而感到烦恼。

(4) 安全：云存储所提供的服务对象范围广泛，因此出现了对服务需求的多样化，有些数据追求可靠，有些数据追求私密，有的数据追求完整，有的数据追求高效率。这就要求云存储能做到满足不同用户不同需求。只有这样，云存储才能被大众所接受。为此，云存储在提供物理隔离的同时，还提供了权限控制。除此之外，云存储还能够对对应的数据进行加密保护<sup>[19]</sup>。为了提高存取效率，云存储还进行了切片处理，这不仅提高了存取速度，在一定程度上还有保密效果，使用这样存储方式，能有效防止非法用户通过单个存储节点获取用户完整的对应信息<sup>[20]</sup>。

## 2.2 ceph 机制

### 2.2.1 集群框架

在 ceph 的框架有以下几个部分组成。

(1) 客户端：通过客户端，用户可以实现根据自己的需求，进行相应的操作。同时，可以通过客户端查看存储系统的健康状态，监控器状态，磁盘使用情况以及存储节点的运行状态等信息<sup>[21]</sup>。

(2) 元数据服务器：用于处理来自客户端传输的一些数据，将这些数据进行编序、切割，形成便于搜索和存储的数据单元<sup>[22]</sup>。同时根据存储节点的存储状况，进行动态重新分数据在磁盘的分布<sup>[23]</sup>，从而提高磁盘的存取性能<sup>[24]</sup>。同时元数据服务器还要和在本存储节点具有相同数据的，其他元数据服务器进行通信。保证了数据副本间元数据服务器的活跃状态。

(3) 对象存储设备：对象存储设备，不仅仅具备存储功能，还具备通信能力，可以实现和其他设备间的通讯。能保证存储相同元素的存储设备间定时检测对方是否正常运作，同时做到让数据在存储设备间进行传输，保证了数据的

可靠性且不占用和外界交互的带宽。

(4) 集群监控器：集群监控器主要功能是监控，当系统中出现有故障的设备，或者当系统中插入了新的存储设备，集群监控器能感知这些变化，并做出相应信息的更新，保证信息表的准确性。

其结构图 2-1 如下。

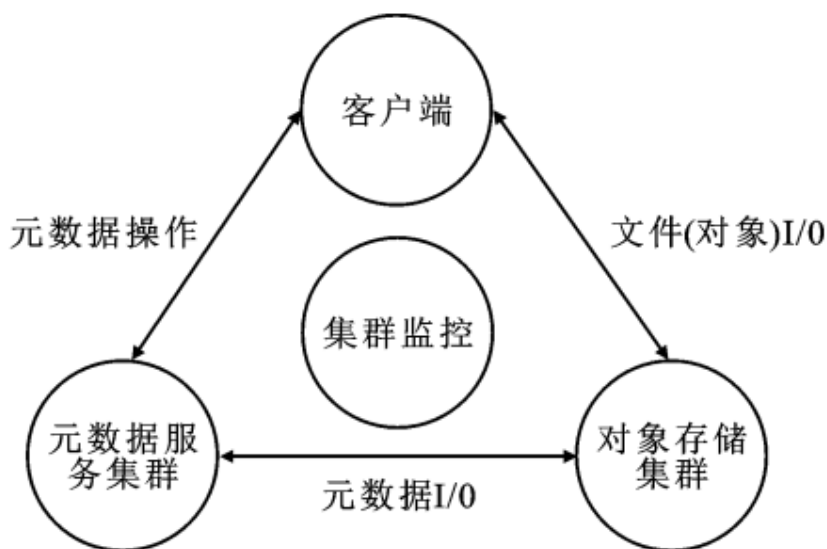


图 2-1 ceph 集群框架

### 2.2.2 存储机制

基于 ceph 的云存储有其独特的存取方式，也正是这种存取方式，使得基于 ceph 的存储系统具有高并发的特性，同时这种存储方式也消除单点故障瓶颈<sup>[25]</sup>。正是因为 ceph 的独特存储机制，使得 ceph 能在众多存储系统中，脱颖而出，并占据很大的市场份额，在今后的分布式存储中，ceph 存储的应用将会越来越流行。

下面将通过宏观的图解去解下，基于 ceph 的存储系统，是如何以其独特方式，进行工作的。

如图 2-2 是 ceph 的存储机制。

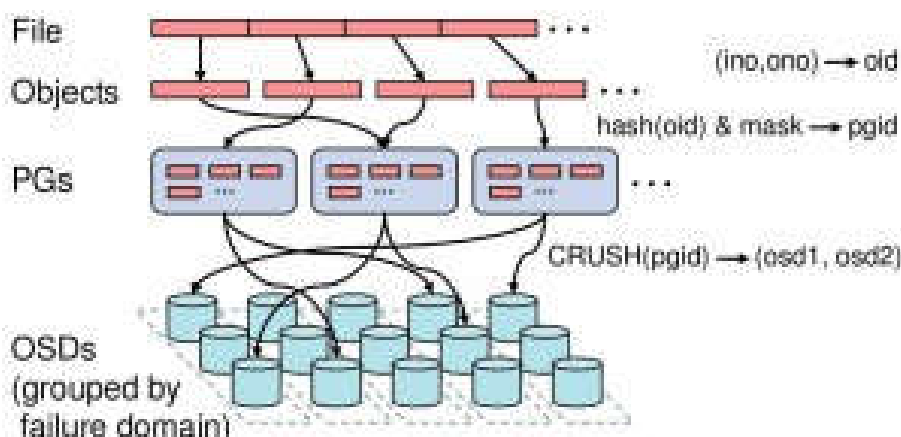


图 2-2 ceph 存储机制

通过图 2.2 我们可以看到，数据的存取操作的实现主要有以下几个部分。

(1) File  $\rightarrow$  Object: 实现方式是给每个文件一个唯一 id，然后再将源文件拆分成事先规定好的、易于在网络中传输的、大小相等的数据块，每个数据块都有自己的编号，用文件 id+块编号就可以在整个磁盘系统中为数据块指定唯一的一个编号，从而将较大文件拆分成多个易于存储的 Objects，方便后续统一管理<sup>[26]</sup>。

(2) Objects  $\rightarrow$  PGs: 将文件分割成诸多 objects 后就需要将这些文件映射到对应的 PGs 上去，映射规则比较简单，可通过公式： $hash(oid) \& mask \rightarrow pgid$  实现对应的映射。其中里面有些限制。之所以采用这种方式，是为了让一个文件所分割的各个部分均匀的放在各个网络磁盘上。为了实现这种情况，要保证 PG 总数为 2 的整数次幂。

(3) PGs  $\rightarrow$  OSDs: 这个过程是将 object 的逻辑组织单元的 PG 映射到实际存储单元 osd 上，采用 CRUSH 算法，将 pgid 带入其中，根据 ceph 对应的配置信息，可以得到有 n 个 osd 组<sup>[27]</sup>。其中 n 既是为了实现可靠性而设置的一个值，代表副本数。在生产环境下通常为 3。

到此，基于 ceph 分布式系统下的低层，完成了从 File  $\rightarrow$  Object  $\rightarrow$  PGs  $\rightarrow$  OSD 整个过程的映射。从整个过程看，我们发现，没有一个地方会涉及到查询全局性的表来获取数据，存取信息的查询任务由客户端完成，从而实现高并发特性<sup>[28]</sup>。这也是 ceph 之所以被很多企业所接受的最重要的一个原因。到此整个 ceph 的存储过程已经完成。



此外，在 `ceph` 上面还有 `cluster map`，在维护磁盘负载均衡方面发挥重要特性，也为磁盘读写性能做出了很大的贡献<sup>[29]</sup>。所以在此讲述下 `cluster map` 的工作机制。随着网络磁盘的使用，磁盘损坏是在所难免的，等出现磁盘损坏时，需要 `cluster map` 去收集、整理信息，网络磁盘数据就会出现数据分布不均衡的现象<sup>[30]</sup>。这就需要通过均衡机制去实现负载均衡。下面讲述 `cluster map` 的特性。

在 `cluster map` 中含有版本号、各个 OSD 的网络地址、各个 OSD 的状态、以及 CRUSH 算法配置参数。其中需要说明的是各个 OSD 的状态。OSD 的状态包括：

`up&in`：该 OSD 处于运行正常状态，且至少承载了一个 PG。

`up&out`：该 OSD 处于运行正常状态，但并未承载任何 PG。

`down&in`：该 OSD 处于发生异常状态，但仍然承载至少一个 PG。

`down&out`：该 OSD 处于发生异常状态，且已经不再承载任何 PG。

其中 `cluster map` 可以收集到这些信息，并以增量形式进行扩散，其中版本号是递增的形式，当出现版本不一致时，以较新的为准。且 `cluster map` 的信息是以异步且 `lazy` 的形式扩散的，从而避免了广播风暴。最终会将 `cluster map` 扩散到全体 OSD 和 `client` 端。各个 OSD 会根据 `cluster map` 进行数据维护，而 `client` 在进行数据存取操作时，需要 `cluster map` 信息，实现自助寻址<sup>[31]</sup>。从而使得基于 `ceph` 的云储存具有高效的存取性能。

通过上面所说的存储机制，我们可以了解到，一个大的文件会被分割成大小相同的、相对较小的数据单元，并被分别存储在不同的 OSD 存储节点上<sup>[32]</sup>。同时，每个较小的数据单元都会被分配到不同的 OSD 存储节点上。这样，同一个文件的不同 OSD 存储节点间会有通信。当 `ceph` 云存储系统中有磁盘出现故障，那么和故障磁盘上具有相同文件的其他 OSD 存储节点，都会检测到这个故障磁盘，当其他 OSD 存储节点检测到这个故障磁盘，就会将这个故障磁盘的信息报告给监控节点。监控节点就会及时更新数据信息。于此同时，当确定故障磁盘已经无法继续使用时，`ceph` 云储存将会启动恢复机制，将故障磁盘所对应的信息，从副本中提取出来，放置在其他正常磁盘上面<sup>[33]</sup>。这样的话就可以做到及时发现故障磁盘，并及时将数据从备份中恢复出来，保证信息的完整性。这些过程不需要人去干预而自动完成。

## 2.3 总结

通过前两节的讲述，可以对基于 `ceph` 的云存储系统的一些概念有了大致的了解。同时通过上面对 `ceph` 存储机制的讲解，我们能看到基于 `ceph` 的云储存，在数据存储方面确实有着不可比拟的优势。`ceph` 云存储不仅实现了高可靠、高性能、高度自动化，更重要的是其解决了单点故障瓶颈问题。由于基于 `ceph` 的云存储运维系统，可同时支持数千个存储节点，实现了大数据对磁盘空间的需求。

## 第三章 需求分析

### 3.1 系统概述

#### 3.1.1 前端概述

合法用户通过验证后，可进入运维中心，在运维中心，选择存储资源模块，可以进入对应的存储资源管理模块。通过界面，用户可以查看不同类型的存储池，根据不同存储池，可以选择性的进入相对应存储池的详细信息界面，查看对应的信息。其中，我所在公司对应的存储池包含两种。分别为 FC SAN 或 IP SAN 和 Server SAN<sup>[34]</sup>。其中 FC SAN 和 IP SAN 内容大致相同，下面分别看下他们对应的详细信息。

**FC SAN:** 在 FC SAN 中包含有存储池的信息，记录着对应 FC SAN 所具备的存储池的个数。每个存储池，又有其磁盘总量和使用情况等信息。在 FC SAN 中，还记录了 VOLUME 的信息，以及 VOLUME 对应的列表信息，在列表信息中，记录了每个 VOLUME 所对应的存储池，以及每个 VOLUME 的大小以及每个 VOLUME 的使用情况，同时还可以在每个 VOLUME 在过去一段时间内的网络速率。在 FC SAN 中还含有磁盘信息，有磁盘总量以及磁盘的使用情况的信息，在 FC SAN 的详细列表中还记录了不同时刻对应的磁盘在过去的一段时间内，磁盘的使用量的信息。

**Server SAN:** 在 Server SAN 下面包含有很多 cluster，而每个 cluster 又有其对应的信息，下面将叙述 cluster 下对应的信息：

**cluster 健康状态:** 这个信息展现了 cluster 的最近一次更新的健康时间点，在 cluster 健康状态栏中，还包含了当前 cluster 健康状态。其中 cluster 的状态包含健康、警告、失败三种状态。当 cluster 的健康状态出现警告或失败时，有对应的产生警告或失败的原因信息。

**监控状态:** 记录着对应 cluster 上含有的所有监控节点，以及监控节点所处的状态。监控节点只含有两种状态，正常和异常。

**IOPS:** 通过折线图，记录了最近一段时间内 cluster 的读写次数的变化情况。

单位为次每秒。

**读写速率：**通过折线图，记录了最近一段时间内 cluster 的读写次数的变化情况。单位为 KB 每秒。

**磁盘总容量：**记录了对应 cluster 总的磁盘大小以及使用情况。

**归置组：**记录了归置组的个数、归置组的状态。其中状态包括三种状态。第一，正常状态，代表这个归置组下面拥有设定的 OSD 个数，且这些 OSD 处于正常运行状态。第二，降级状态，当对应某个归置组下运行的 OSD 个数，没有达到预先规定的 OSD 个数，或者某个归置组下对应的 OSD 出现了故障，没能正常运行起来，此时的归置组处于降级状态。第三，当一个归置组下对应的 OSD 全部出现故障或此归置组下没有指定 OSD，此时的归置组处于异常状态。

**OSD 存储节点：**OSD 存储节点也就是数据存储的单元。每个机架上会有很多台存储服务器，每台服务器上会拥有很多个 OSD 存储节点，每个 OSD 存储节点记录了本 OSD 存储节点的 ID 编号，OSD 存储节点所属的主机，OSD 存储节点的状态，OSD 存储节点的容量信息，OSD 存储节点已用空间信息，每个 OSD 存储节点所对应的 IP 地址及对应的端口号和编号，通过运维界面，还可以控制 OSD 存储节点的状态。

OSD 存储节点是整个云储存的核心，OSD 存储节点的存储状态直接影响到整个存储系统的性能。其中，关于云存储系统的负载均衡，其实就是指在各个 OSD 存储节点上的数据是否处于良好的平衡存储状态。在 OSD 界面会有整个 cluster 的磁盘使用偏差值信息，可以依据这个信息，决定是否进行手动均衡。为了达到很好的用户体验，可通过均衡数据在云存储磁盘系统中的分布来实现。系统中还提供了自动均衡的功能，可以给系统设定每周的哪几天，在哪个时间段，设定最大偏差值为某个具体值，让系统自动做均衡操作。这样可以做到不影响用户正常操作的情况下，调整 OSD 存储中的数据，从而实现数据在 OSD 中负载均衡。

**POOL：**每个 POOL 的详细列表里面，详细介绍了每个 POOL 所对应的 ID，每个 POOL 所拥有的副本数，每个 POOL 所对应的归置组个数，每个 POOL 最大容量以及已使用容量，每个 POOL 还记录了其在一段时间内，对应的 IOPS 数据。通过用户界面，用户还可以设置每个 POOL 对应的归置组个数和对应 POOL 最大的空间容量。

### 3.1.2 后端概述

此处所说的后端包括后面的 Java 代码，和 Java 交互的 Calamari 部分代码，和 Calamari 对接的 ceph 系统以及和 ceph 系统交互的 Centos 服务器系统。

用户利用前端交互界面，将对应的请求，通过后台控制器（angularjs）实现和后台 Java 代码的交互，Java 代码部分对用户发出的请求进行封装，通过调用服务器上 calamari 对应的 API 实现和后台数据库的交互，后台 calamari 是对底层 ceph 的近一步封装，提供给上层监控信息，同时将上层的命令传达给下面 ceph 层，下面 ceph 层收到来自 calamari 的信息后，执行对应的操作，并将对应的数据发送给 calamari，由 calamari 将数据转交给上面的 Java 层，Java 层将得到的信息传送给前台 angularjs 控制器，再有前台 angularjs 控制器实现前台信息的展示。

### 3.1.3 系统构造

通过前两部分，我们已经大致了解到系统组成要素，以及各个组成要素的含义，下面将通过 Server SAN 的实例，结合结构图将系统的整体结构展现出来。下面将分两部分展现系统结构图。

由系统进入 Server SAN 的结构图如图 3-1 所示。

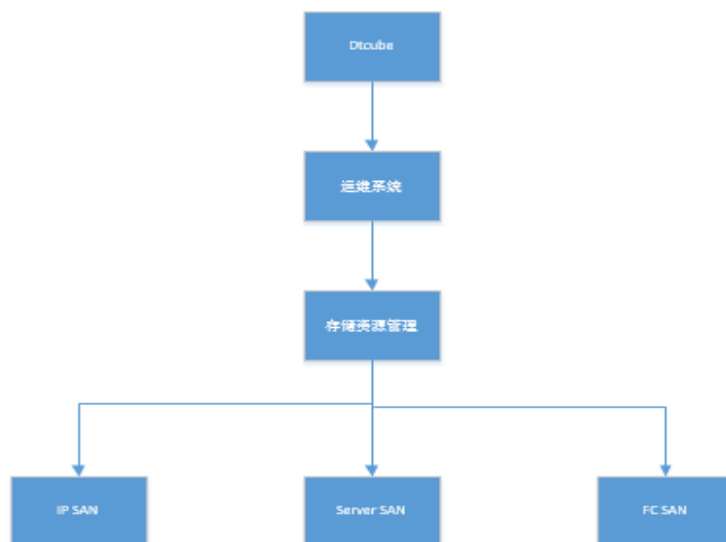


图 3-1 系统结构图

下面图 3-2 是在 Server SAN 结构内部对应的结构图。

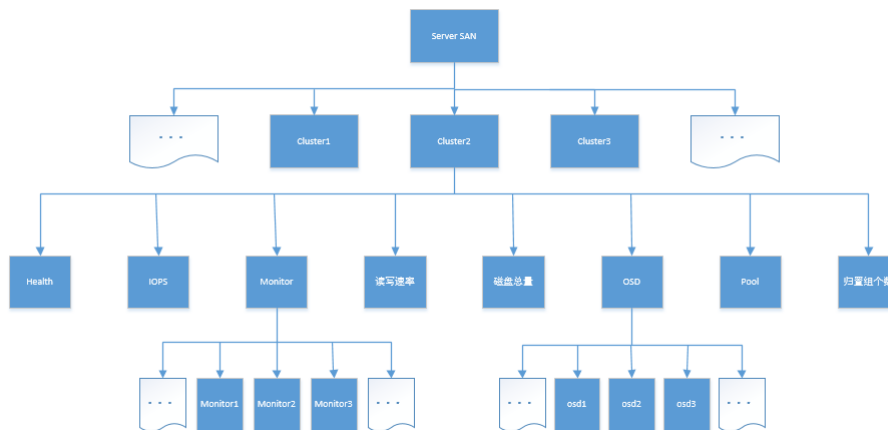


图 3-2 Server SAN 内部结构图

### 3.2 系统总体架构

此处讲述下基于 ceph 的云存储运维系统的总体架构。最底层是存储介质设备，所有的一切操作，最终都是对这层存储介质进行操作的。在存储介质上，我们安装了 Centos 后台服务系统，用来操纵最终的存储介质。直接和 Centos 系统交互的是部署在 Centos 系统上的 ceph 存储系统的 RADOS 层，在 ceph 存储系统上层，对 RADOS 进行了抽象，出现了 LIBRADOS 层，LIBRADOS 是对 ceph 存储系统 RADOS 层的抽象和封装，并向上提供 API，从而实现更好的基于 RADOS 的，对 Centos 服务系统的操作。在 ceph 存储系统中，LIBRADOS 层之上还有进一步抽象，其中包括 RADOS GW、RBD、CEPH FS 他们是更高层次的抽象和封装，根据实际需求而选择不同封装层次和类别。与 ceph 存储系统直接交流的是上层的 Calamari 层，Calamari 层负责将上面下发的命令，提取相应参数，在调用对应的 API 接口，实现将上层所需要的工作。和 Calamari 有交互的是后端的 Java 代码实现部分，在 Java 部分采用了 spring MVC 框架和 hibernate 框架，实现了用户界面对后台的相应操作。和 Java 部分对接的是前端的 JavaScript 语言，我们采用了 Angular JS 框架，很好的实现了用户间的各种

需求，数据的动态绑定。在和后端服务器交互时，有对应的封装去实现和后台的交互，极大的简化了繁琐的后台连接代码。**Angular JS** 又是前端用户界面的控制器，实现了双向数据绑定，方便用户和系统的交互。同时，也使得前端界面开发起来更加方便、便捷。业务的整体流程图如图 3-3 所示。

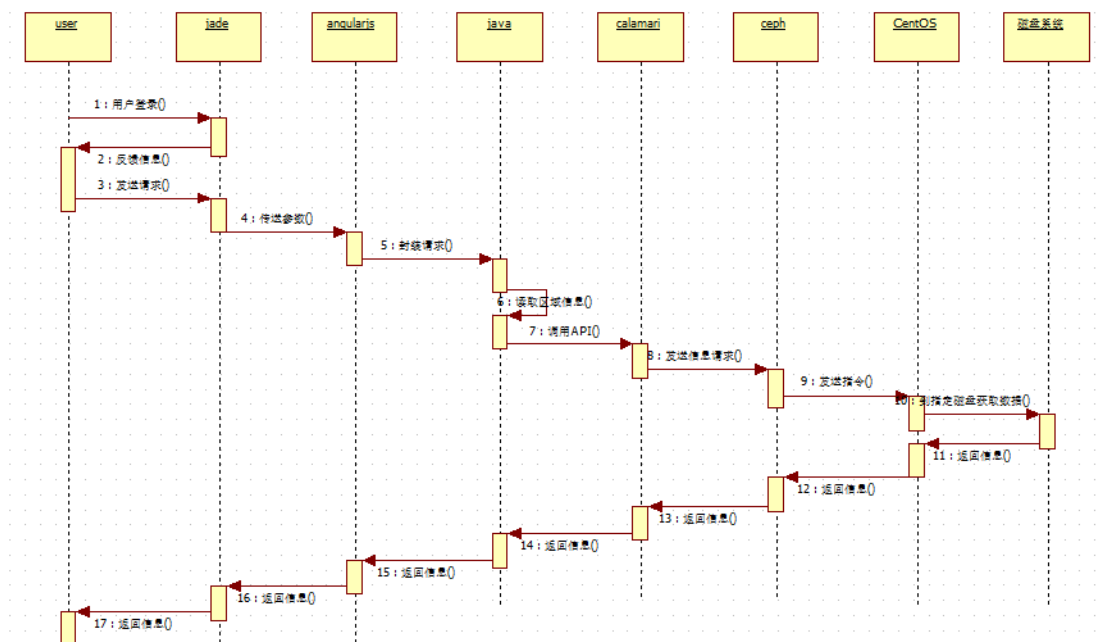


图 3-3 业务请求的总体流程

### 3.3 云存储运维系统性能需求分析

随着云计算和物联网的兴起，数字数据信息急剧增加。公司、企业、政府等部门将会产生极大的数据信息，他们需要有良好存储介质去保存这些数据，同时，他们对不同信息有不同要求。他们有各式各样的数据存储需求，下面将着重讲述，他们对云存储数据的需求。

### 3.3.1 高性能需求

ceph 存储机制中，消除单点故障机制，实现了所有数据的存储和读取，将不再必须通过查找某些固定的信息表就可以实现。即实现了 client 和 server 的直接通信，在其通信过程中摆脱了中间代理和相应的信息转发。

通过上面讲述的 ceph 存储机制，我们可以知道，一个原文件在存储前是被分割成很多大小相同的 object，且几乎所有的 object 都被分配到不同的 OSD 存储节点上，这样就可以实现对某个文件操作时，可以使得所有含有本文件的 OSD 存储节点同时工作，实现了存储服务器的高并发的特性。

用于存储数据的每个 OSD 存储节点，都有对应的权重值，参照权重值，考虑是否进行负载均衡，从而更充分发挥每个 OSD 存储节点的性能。

保证数据不会因一个地方的磁盘损坏，而造成数据丢失的最直接的方式，就是异地创建副本。创建副本就需要考虑一个问题：创建副本的过程中，对宽带的的影响。在基于 ceph 的云存储系统中，client 只需要和标记为 primary 的磁盘进行数据传输就可以了。在有标记为 primary 的 OSD 存储节点通过内部存储网，创建预先规定的对应的副本，这样就可以避免有 client 去创建副本而造成的网络消耗，提高了存储性能，同时保证了数据信息的安全性。

### 3.3.2 高可用性需求

在 ceph 存储系统中可以通过配置 per-pool 来设定每个文件分割成 object 对应的副本数，同时还可以设定故障域，保证数据的各个副本不在同一个地方，使得数据安全系数更高。

因基于 ceph 的存储系统可以实现 client 和 server 端的直接通信，消除了单点故障隐患，允许系统出现很多种故障情形。这种机制还可以支持单个组件滚动升级和在线替换。通过 ceph 存储机制，我们可以了解到，同一个文件被别存储的不同的 OSD 存储节点上，这些 OSD 存储节点间会相互通信。当某个 OSD 存储节点出现了故障，与其通信的相邻 OSD 存储节点就在一个通信周期内，会感知到故障 OSD 存储节点，并将这种情况汇报给监控节点，使得当 OSD 存储节点和监控节点交互时，能及时得知系统中 OSD 存储节点状态。同时会激发 ceph 存储系统自动恢复机制，并将最终恢复后的状态反馈给监控器和与其相通信的其他 OSD 存储节点。整个恢复的过程中，不需要人工干预。在 OSD 存储



节点恢复过程中，仍然可以通过访问故障 OSD 存储节点上的对应数据的备份 OSD 存储节点上的数据，实现数据正常的访问。当数据出现意外时，如果需要恢复，因数据是被分割成很多大小相同的 object 块，存储在不同的 OSD 存储节点上，当要恢复数据时，多个 OSD 存储节点可以实现同时对数据的恢复操作，极大的降低了数据的恢复时间，更好的提高了用户体验<sup>[35]</sup>。

### 3.3.3 高扩展性需求

因为在 ceph 存储系统中，消除了中心节点这个瓶颈，将对应的任务划分到了各个存储节点，使得每个存储节点具备更多的处理能力，从而可以实现高度并行。

由于 ceph 存储系统具有在线扩展、替换、升级的功能。当存储介质发生故障时，可以实现数据的自动恢复，当系统中出现了磁盘添加或删除时，ceph 内部均衡机制算法会重新计算数据存储操作，实现数据的均匀分布，不在需要人工去干预。使得对固件的删除、增添和固件的升级不会在影响正常的业务操作。

## 3.4 云存储运维系统功能需求分析

系统要保证能时刻去检测更新对应磁盘的信息，并将这些信息及时返回到用户界面。其中需要检测的信息包含如下：（1） 集群列表；（2） 集群的整体健康状况；（3） 监控状态；（4） IOPS 性能；（5） 读写性能；（6） 磁盘容量以及使用了情况；（7） 归置组个数和状态；（8） 存储池的个数和状态；（9） OSD 的个数和状态。

通过这些信息可以大致了解基于 ceph 云存储系统下，磁盘数据存储情况，以及整个云存储系统运行情况。在底层的系统中，当出现磁盘故障或去除过胜磁盘或者对部分硬件升级，云存储系统可以自动完成这些工作，且对用户的操作影响不大。

以上主要是讲述云存储系统的监控功能需求，其中还有一个较重要的需求，即数据的均衡操作。当磁盘数据出现分布极端化时，会严重影响到用户数据的存取性能操作，使得系统的整体运行性能有很大的下降。所以系统提供了均衡机制，因均衡机制使得数据会出现大规模的迁移，占用系统资源较大，均衡期

间会严重影响到用户的操作效率。所以在系统中提供了两种均衡机制：自动均衡和手动均衡。当磁盘使用偏差值超过正常范围，通过自动均衡机制，可以按用户设定的日期和时间点去做均衡操作。这样可以做到最大限度的减小均衡时，对用户业务的影响。手动均衡操作，是用户参照当前数据磁盘存储偏差，手动执行均衡操作。这个操作会使得系统立刻做均衡操作，而此时如果有用户在做磁盘操作，这时对用户正常操作影响较大，应慎重使用。

## 3.5 总结

通过本章节的讲述，可以对基于 `ceph` 的云存储运维系统在功能和结构上有大致的了解。前几节尽可能详尽的介绍了云存储运维系统中所涉及到功能和实现机制，下面将更加深入的去讲解实现过程。

## 第四章 系统设计

### 4.1 系统架构设计

整个系统的设计有好几个部分组成，为了以后升级和维护的需要，在设计的过程中会考虑让每个层次之间尽可能的松耦合。后期随着用户需求的不断增加或系统升级的需要，整个系统的维护只需要改动局部实现逻辑就可以达到用户需求或系统升级的需求。整个系统的设计架构如下：

#### 4.1.1 前端架构设计

前端是采用 nodejs、npm、bower、gulp 搭建，这些工具为前端开发提供了极大地便利。下面将分别介绍这些工具在系统架构中如何构建前端架构。

(1) **nodejs**: nodejs 其实是一个 JavaScript 运行环境，是对 V8 引擎进行了封装，提供了相应的 API 共，方便用户进行调用。

(2) **npm**: 在安装 nodejs 的过程中，会附带安装上 npm，npm 是 nodejs 程序包的管理工具，根据需求，实现从网上下载所需要的对应的安装包。同时还可以管理在项目中包的依赖关系，用于管理项目所需程序包

(3) **bower**: bower 是个包管理器，用于管理项目中对 js 类库的依赖。至于要一条命令就可以将对应的依赖库下载到本地对应的位置。是应用于 web 应用的包管理器。

(4) **gulp**: gulp 是用于前端构建的工具，用于检测文件变化，并按设定的参数，将变化的内容写到对应文件中，压缩 js 文件，启动 web 服务。

这是前端所涉及到的工具，为前端开发提供了极大的便捷，通过这些工具，只要将所需要的类库写到对应文件，再执行相应命令就会自动下载，并添加到指定位置，为以后的扩展提供极大便利。

### 4.1.2 后端架构设计

后端包含好几个部分，通过这几个部分的整合，构成了项目的整个后端，同时，这几个部分也使得整个项目在解耦方面做得比较好。下面将讨论后端代码的实现。

首先是和前台控制器交互的是 Java 代码，在 Java 代码里使用了 springMVC 框架，对前端发出的请求进一步封装，传输给了更下层的 calamari 模块，calamari 是 ceph 的一个管理平台，相当于对 ceph 做了进一步的封装，对外提供了一套具有特殊需求的 REST API 接口。当 calamari 接收到上层传输的请求后，再向 ceph 层发送相应请求，ceph 根据具体请求，获取对应数据，然后将这些数据传送到上层的 calamari 模块，再由 calamari 模块将数据传输给 Java 代码模块，在 Java 代码块，将数据做相应的封装，然后传送给前端控制器，从而实现对相应数据信息的请求。

## 4.2 云存储底层设计

先讲述下，在实际开发中，用作底层部署的 ceph 的环境。实际开发、测试中，用到的服务器是惠普服务器，用三台做环境测试，每台服务器上拥有 12 块磁盘。在裸机情况下，可通过 PXE (preboot execute environment, 预启动执行环境) 装机。装机时选用软 RAID 进行装机。通过这种装机方式，可以实现将 sda 和 sdb 都用作系统盘。装机后的磁盘分布情况如下：

```
[root@localhost~]# lsblk
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
sda	8:0	0	1.8T	0	disk	
└─sda1	8:1	0	500M	0	part	/boot
└─sda2	8:2	0	31.3G	0	part	[SWAP]
└─sda3	8:3	0	19.5G	0	part	/var/log
└─sda4	8:4	0	1K	0	part	
└─sda5	8:5	0	1000M	0	part	/home
└─sda6	8:6	0	1.8T	0	part	/
sdb	8:16	0	1.8T	0	disk	

```

└─sdb1  8:17  0   500M   0 part /boot
└─sdb2  8:18  0  31.3G   0 part [SWAP]
└─sdb3  8:19  0  19.5G   0 part /var/log
└─sdb4  8:20  0   1K      0 part
└─sdb5  8:21  0 1000M   0 part /home
└─sdb6  8:22  0   1.8T   0 part /
sdc      8:32  0   1.8T   0 disk
sdd      8:48  0   1.8T   0 disk
sde      8:64  0   1.8T   0 disk
sdf      8:80  0   1.8T   0 disk
sdg      8:96  0   1.8T   0 disk
sdh      8:112 0   1.8T   0 disk
sdi      8:128 0   1.8T   0 disk
sdj      8:144 0   1.8T   0 disk
sdk      8:160 0 447.1G  0 disk
sdl      8:176 0 447.1G  0 disk

```

其中 sdc~sdj 磁盘为 SATA 磁盘，用作数据存储磁盘，sdk 和 sdl 为 SSD 磁盘，用作 journal 磁盘。同时，在每台服务器上部署一个监控节点。服务器间的数据传输通过内网实现。因为在实际工作中，环境的部署是通过对应的集成脚本去部署的，所以为了更好的讲述细节，我又在自己的机器上，做了一个 ceph 部署安装。

下面通过讲述我在自己机器上部署的一个 ceph 集群环境，来大致了解下，云存储底层的大致设计理念。使用的系统是 Ubuntu16.04 版本。大致涵盖以下几个内容：在 Ubuntu16.04 下安装了 virtual box 虚拟机，通过虚拟机安装对应的服务系统；在虚拟机上尝试了手动部署 ceph 集群；尝试了在 virtuabox 虚拟机下建立 7 个 centos 7.2 服务器系统，其中一台服务器作为安装集群的主服务器，剩下的六台服务器，其中三台服务器做 OSD 服务器，剩下的 3 台服务器做 MON 服务器，具体设计理念如下：

除主服务器和监控服务器外，为其他三台服务器配置两个网段，一个公共网，用于和外部做信息传输，还有一个用于内部数据做副本时，传输数据的

内部网络。这样可以保证用户数据在做备份时，不会影响到外界数据信息的传输速率。

为了更贴近现实中 ceph 存储系统的部署，我们给 OSD 服务器添加了额外的四块磁盘，加上创建 centos 服务器时的系统磁盘，在 OSD 服务器上有 5 块磁盘，其中一个做系统盘，安装时分配了 20GB 的存储空间，后面几块是安装好系统后手动添加的几块磁盘，其中有一块做日志磁盘，对空间需求不大，剩下又添加三块存储空间较大且相等的磁盘，用作存储数据。同时对用作日志的磁盘进行空间划分，将空间划分为三部分，用来对应每台 OSD 服务器上的各个数据盘。

其中三台 MON 服务器可以通过公共网络监测系统中 OSD 服务器中每个 OSD 状态，并将这些信息记录在 primary 监控器中。于此同时 primary 监控器所获的数据会生成两副本，分别存储到另外两个服务器中。防止出现单台 MON 服务器出现事故，而造成数据丢失现象。安装了三台 OSD 服务器，也是为了使每个存储文件能有三副本。正常生产环境中，OSD 服务器的数量远远超过三台，在此只是说明了 ceph 存储系统中的一些特点。通过这种设计，可以领略下在自己机器上部署 ceph 集群所需要考虑的问题，以及 ceph 集群中要注意的事项和一些基本原理。

### 4.3 总结

通过本章的讲述，我们了解了基于 ceph 云存储运维系统中所涉及到的一些技术和工具以及云存储底层的一些基本情况。本章概述了在实际开发中所用到的环境，自己为了实现 ceph 的部署而对应的实验环境。在下面章节将更加具体的介绍系统在实际开发中和自己实验中的设计和实现，以及在底层 ceph 云存储是如何部署、安装的。

## 第五章 系统的部署及管理

### 5.1 云存储环境搭建和部署

通过实际工作中所用到的框架，结合自己的理解，大致讲述下，项目中所涉及到的系统架构。此处所述的框架，更倾向于独立讲述各个模块所涉及到的内容。在整个项目开始前，需要将项目所运行的环境搭建起来，接下来将讲述项目所涉及到的所有环节环境的搭建。

#### 5.1.1 ceph 存储系统的搭建和部署

先看下在实际的工作中，部署 ceph 的过程。通过上一章可以了解到，在实际的工作中，我们使用了三台惠普服务器，通过 PXE 对三台惠普服务器进行了装机，在装机的过程中，安装了 dtcube 的集成脚本。下面来看下，在实际工作中，ceph 环境的部署过程。

（1）配置服务器的管理网：通过在三台服务器上分别执行命令：  
`dtcube-deployer config-ip eno1 10.158.113.200/201/202 255.255.255.0 --gw 10.158.113.1` 实现对服务器的管理网的配置。其中 eno1 是各个服务器所在物理机的初始管理网的接口，10.158.113.200/201/202 是初始管理网的 IP，255.255.255.0 是初始管理网的掩码，--gw 为对应的管理网网关。

（2）格式化磁盘：通过执行下面的命令，实现对磁盘的格式化：  
`dtcube-deployer format-disk "sdc sdd sde sdf sdg sdh sdi sdj sdk sdl"`。因为在进行 PXE 装机时，选择的是软 RAID 模式，所以“sdb”盘也被用作了系统盘，所以在格式化时，不能格式化“sdb”盘。

（3）启动部署节点：选择一台服务器作为部署的虚拟机，并通过命令：  
`dtcube-deployer create eno1 10.158.113.223 --gw 10.158.113.1` 实现部署虚机的设置。其中 eno1 是所在物理机的初始管理网的接口，10.158.113.223 为分配给部署虚拟机使用的部署 IP 地址，--gw 是部署虚拟机对应的网关。

（4）部署虚拟机登录：启动部署虚拟机后，就可以在浏览器上输入 <http://10.158.113.223> 进入部署界面，其中 <http://>后面的 IP 就是启动部署节点所分配的 IP 地址。输入网址后会进入如图 5-1 界面。



图 5-1 dtcube 登录界面

（5）添加主机：在云平台界面，点击：部署—>主机管理—>添加，实现对主机的添加。其中添加主机界面如图 5-2 所示。



图 5-2 添加主机界面



点击“添加”后会出现出创建主机界面，里面含有创建主机的信息，如图 5-3 所示。

创建主机

\*主机名称

以字母开头包含字母、数字或短横线

\*IP

IP须满足 "XXX.XXX.XXX.XXX" 格式

\*位置

格式为 "机架号:所在槽位/总槽位数"

描述

确认

取消

图 5-3 主机参数界面

在主机参数的对话框中，给主机起对应的名称，服务器 IP 地址，根据实际情况，填写主机所在服务器的位置，并添加对应的描述信息。IP 地址为配置服务器管理网所对应每台服务器的 IP 地址。通过这个功能，将对应的三台服务器添加进来。添加结果如图 5-4 所示。

<input type="checkbox"/>	主机名称	IP地址	所属集群		部署状态	运行状态	位置	描述信息	操作
<input type="checkbox"/>	compute01	10.158.113.100	-		未部署	运行中	1.1/10	huawei-compute	<a href="#">操作</a>
<input type="checkbox"/>	compute02	10.158.113.101	-		未部署	运行中	1.2/10	huawei-compute	<a href="#">操作</a>
<input type="checkbox"/>	compute03	10.158.113.102	-		未部署	运行中	1.3/10	huawei-compute	<a href="#">操作</a>

图 5-4 主机信息

（6）创建集群：在云平台界面，点击：部署—>区域管理—>创建，实现对集群的创建。如下图 5-5 所示。



图 5-5 部署界面

在创建的对话框，会涉及到集群的名称和相应的描述。如下图 5-6 为创建界面信息。



图 5-6 集群创建

填写对应项，点击确定即可实现集群的创建。对应集群如图 5-7 所示。

集群名称	主机列表	部署状态	描述	操作
cube-release	server-1, server-4, server-2, server-3	已部署	—	操作

图 5-7 集群实例

通过点击“操作”选项，会有对应管理主机选项。如果 5-8 所示。



图 5-8 操作界面

选择添加主机选项， 如下图 5-9 所示。



图 5-9 主机添加界面

选中对应主机，点击确认，即可添加到刚刚创建的集群中去。

（7）部署：通过界面“部署”按钮，进入部署界面，如图 5-10 所示。



图 5-10 部署界面

点击“部署”按钮，开始部署，根据提示会弹出对应的信息，如果 5-11 是配置基本信息的界面。

部署 dt-test

基础配置

名称: dt-test

存储方案: ☒ Server SAN ☒ IP SAN

网络配置

主机配置

安全配置

下一步

5-11 参数界面

可通过选项进相对值的设定。此处存储方案,选择的是 Server SAN 和 IP SAN 两种存储模式。点击“下一步”，进入存储方案的配置，根据界面提示，进行相应选项的设定，如图 5-12 所示。

存储配置

Server SAN 配置

型号: High-Performance

故障域范围: ☒ 主机 ☐ 机架

IP SAN 配置

名称: dell

生产厂商: DELL

用户名: admin

管理地址: 10.158.111.100

存储方案: ☒ Server SAN ☐ NFS

添加 删除

上一步 下一步

图 5-12 存储方案

需要说明下，型号对应三种类型，分别对应为：Cost-Effective、High-Performance、Versatile 三种类型。下面分别描述下，这三种类型的关系和区别。

**Cost-Effective:** 需要 4 块磁盘；其中槽位 1 的磁盘做系统盘；槽位 2-4 的磁盘做数据盘。

**High-Performance:** 需要 12 块磁盘，其中槽位 1-2 的磁盘做系统盘（可以将槽位 1 单独做系统盘，也可以将槽位 1 和槽位 2 形成软 raid，做系统盘）；槽位 3-10 的磁盘做数据盘；槽位 11-12 的磁盘做 journal 盘。

**Versatile:** 需要 7 块磁盘，其中槽位 1-2 的磁盘做系统盘（可以将槽位 1 单独做系统盘，也可以将槽位 1 和槽位 2 形成软 raid，做系统盘）；槽位 3-6 的磁盘做数据盘；槽位 11 的磁盘做 journal 盘。

通过三种信息，结合我们服务器的配置，此处应该选择 High-Performance 这种类型。在 IP SAN 处的用户名和密码对应后端 IP SAN 的登录信息。设置管理 IP SAN 的对应 IP。点击下一步，进入网络参数的配置，详细参数选项如图 5-13 所示。



图 5-13 网络参数配置

控制台访问地址是部署在控制单元上的 dtcube 的访问地址，是管理网段地址，但是不能在下面分配的管理网段起始范围内。初始管理网即部署网络。复用初始管理网 IP，如图 5-13，则需要分配至少 40 个 IP 地址。不复用初始管理

网 IP，则需要新建独立管理网，新建的管理网则需要分配至少 72 个 IP 地址，初始管理网只需要预留 3 个 IP 地址即可。网关指的是管理网网关。存储网的网段里，至少分配 72 个存储 IP 地址。这 72 个 IP 地址除了作用于各个节点的 IP 配置，还预留了部分作用于业务虚拟机使用。VLAN tag 是在接口复用的情况下用，而逻辑 IP 不复用的情况下，需要根据实际组网环境分配 VLAN tag 给不同的网络，其他情况则视具体情况而定。数据网是分配给 VM 的 VLAN 范围，供内部数据传输数据。

填写完对应信息，点击下一步，进入配置主机接口信息栏。在配置主机界面，会有每个主机对应的信息。在每行信息最后有个“操作”选项，点击“操作”选项，做相应网卡配置。网卡详细信息如图 5-14 所示。

网卡详情

网卡	网口	状态	速率	MAC地址
Intel Corporation I350 Gigabit Network Connection (rev 01)	eno1	UP	1000Mb/s	8c:dc:d4:af:35:a8
	eno3	DOWN	Unknown!	8c:dc:d4:af:35:aa
	eno4	DOWN	Unknown!	8c:dc:d4:af:35:ab
	eno2	DOWN	Unknown!	8c:dc:d4:af:35:a9
Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)	ens1f0	UP	10000Mb/s	6c:92:bf:13:95:96
	ens1f1	DOWN	Unknown!	6c:92:bf:13:95:97

创建逻辑绑定网口

添加

管理网

eno1

存储网

--

数据网

--

确认

返回

5-14 网卡信息图

如图 5-14 所示，为存储网和数据网分配万兆网卡。点击确定实现网卡信息的配置。为每台服务器配置对应的网卡。如下图 5-15 所示。

主机名称	备注	初始管理地址	初始管理接口	管理接口	存储接口	数据接口	操作
compute01	huawei-compute	10.158.113.100	ens1	ens1	ens1f1	ens1f1	<a href="#">分配网卡</a>

5-14 配置网卡的主机

执行下一步，进入执行部署界面，点击“部署集群”，即可开始部署集群。  
在部署集群的过程中，有进度信息，如图 5-15 所示。

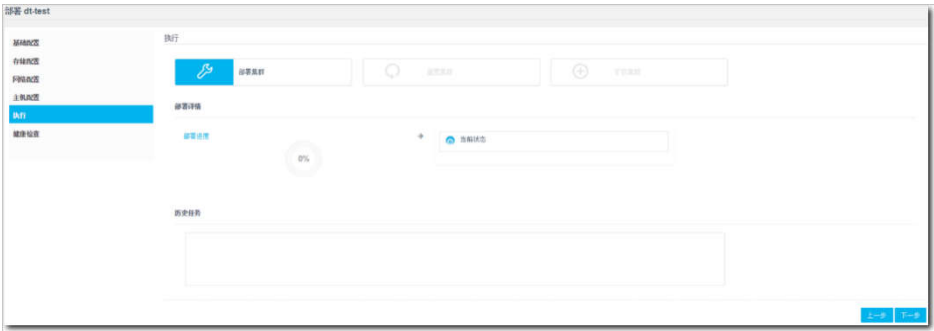


图 5-15 部署执行界面

当进度条走到百分之百后即可完成部署工作。部署完成后，点击“跳转到已部署集群中”，实现部署节点到控制节点的跳转。部署节点将会关闭。如下图 5-16 所示。



图 5-16 部署完成界面

到此，ceph 的部署已经全部完成，所有的细节都被集成到了脚本。通过上面的讲解我们了解了在实际开发中所涉及到的东西，接下来通过我在本机上的部署，展示 ceph 部署的更多细节。

在本机上，手动搭建 ceph 分布式存储集群大致需要下面几部。

(1) 集群配置方案：在示例中使用三台服务器做 MON 服务器，此处说的说的服务器是虚拟机服务器。配置为：centos 7.2，1 个 CPU，1GB RAM，只需一个 20GB 的磁盘做系统盘就可以了，磁盘分区格式为 XFS。对于 OSD 服务器，需要三个 OSD Daemon 用于存储条带化的数据。这种需求可以通过在三台 OSD 服务器上，为每台 OSD 服务器，安装三个 OSD Daemon，这样方便实现数据的分块存储和三副本备份。同时，由于 OSD 服务器需要做数据备份，为了避免公网宽带紧张，可以建立一个内部局域网，从而实现在数据备份时不影响其他公网多数据的操作。具体服务器的列表如表 5-1 所示。

表 5-1 服务器网络配置

主机名	公网	公网子网掩码	集群内网	内网子网掩码
admim	192.168.100.1	255.255.255.0	——	——
mon1	192.168.100.11	255.255.255.0	——	——
mon2	192.168.100.12	255.255.255.0	——	——
mon3	192.168.100.13	255.255.255.0	——	——
osd1	192.168.100.21	255.255.255.0	192.168.101.21	255.255.255.0
osd2	192.168.100.22	255.255.255.0	192.168.101.22	255.255.255.0
osd3	192.168.100.23	255.255.255.0	192.168.101.23	255.255.255.0

其中 OSD 服务器需要特殊说明下。在所有虚拟服务器中，admin、mon1、mon2、mon3 服务器都只需要一个系统磁盘就可以满足基本需求。为了能更好的做到数据分块存储、数据三副本备份，需要在 OSD 上额外添加四块磁盘，安装时的磁盘用作系统盘，新挂载的四块磁盘中，一块做日志磁盘，保证数据操作的一致性，另外三块磁盘用作数据磁盘，用于存储数据和做数据的备份。

(2) 安装配置 OS：演示中，所有的服务器都是安装 centos 7.2，安装类型选择了 minimal install，使用 root 用户对服务器进行操作。为了保证服务器间的时间一致，在除 amin 服务器意外的所有服务器上安装了 ntp 服务器。修改 /etc/ntp.conf 下的文件，添加 server 0.cn.pool.ntp.org; server 1.asia.pool.ntp.org;



server 2.asia.pool.ntp.org 用于获取中国区公用时间，同步服务器。在试验中为了能更好的运行 ceph 分布式系统，执行了关闭防火墙操作。通过命令：`systemctl disable firewalld`；`systemctl disable firewalld` 实现防火墙的关闭操作。为了更好地通信，还需禁用 SELinux：将 `/etc/sysconfig/selinux` 中的 SELINUX 的状态由 enforcing 变成 disabled 状态。为了防止在安装集群的各个节点时报错，需要注销掉 `/etc/sudoers` 中的 `requiretty` 选项。只在 admin 机器上进行 ssh 设置，修改 `/etc/hosts` 配置文件，将公网 IP 地址和对应的服务器名称相匹配，通过 `ssh-keygen -t rsa` 生成 ssh key，通过命令：`ssh-copy-id {服务器地址}`，将生成的对应的公钥复制到其他服务器上。编辑 `~/.ssh/config` 文件，为每台服务器分配用户名和密码，从而可以实现 admin 到各个服务器无需密码即可登录、操纵各个服务器。

(3) OSD 硬盘分区：通过前面的系统安装可知，每个 OSD 服务器都装有 5 个磁盘：系统盘 1 个 (sda)，日志盘 1 个 (sdb)，数据盘 3 个 (sdc、sdd、sde)。下面进行磁盘分区：a) 系统盘：保持不动，安装时自动做了相应分区和格式化。b) 日志盘：将日志盘分为三个分区，给每个分区分配整个磁盘的 33%。格式无特殊要求。c) 数据盘：每个磁盘用作一个分区，且将磁盘格式化为 xfs。通过 parted 工具实现对磁盘的分区和和格式化。执行完上述步骤，我们会发现 sdb 下有三个分区，分别为：sdb1、sdb2、sdb3，到此完成了日志磁盘的划分和格式化。用同样的方式，利用 parted 工具，对数据盘进行分区和格式化。为三个 OSD 服务器上的磁盘都做相应的分区和格式化。到此，磁盘的分区和格式化已全部完成。

(4) 安装 ceph：a) 在安装 ceph 之前需要导入 key；b) 建立 ceph 的 yum 源，创建并编辑 `/etc/yum.repos.d/ceph.repo` 文件，以便后期获取对应的软件；c) 通过 yum 安装插件 `yum-plugin-prioriti`；d) 安装依赖包，ceph 的安装需要依赖 snappy、leveldb、gdisk、python-argparse、gperftllos-libs 等第三方软件包。有些软件包不在系统当前 yum 源中，如果要安装这些软件包，需要安装 EPEL 的 yum 源。安装 EPEL 的 yum 源，执行命令：`yum -y install epel-release`。执行完命令后需要手动修改对应生成的 `/etc/yum.repos.d/epel.repo` 和 `/etc/yum.repos.d/epel-testing.repo` 文件两个文件，将文件中 `baseurl` 替换成：`http://mirrors.neusoft.edu.cn/epel/7`，同时需要注释掉 `mirrorlist` 选项。这是为了在国内能更好的链接安装软件所需要的软件包；e) 清除缓存并更新源，通过命令 `yum clean all` 和 `yum update` 实现清除原有的 yum 源，并更新最新定义的 yum 源；

f) 安装第三方软件包，通过 yum 命令，实现对 snappy、leveldb、gdisk、python-argparse、gperftools-libs 的安装。这个这条命令可以将 ceph 依赖的软件包都安装到服务器上，从而使得服务器能够正常安装 ceph；g) 安装 ceph，通过 yum 命令实现对 ceph 的安装。

到此，ceph 的基本安装就完成了，下面将借助 ceph 这个工具去部署 MON 节点和 OSD 节点。

(1) 部署 MON 节点：MON 节点用于监控整个运维系统的状态，为告知每个新添加的 OSD 存储节点，关于整个系统中集群信息状态。在运维系统中至关重要。在做 MON 节点部署前，需要去了解一些关于 MON 节点的一些基本概念，后面的部署会用到这些概念性的东西。a) fsid：每一个集群拥有一个 fsid，且每个集群的 fsid 唯一，用于识别不同集群。b) cluster name：集群名字，为集群起一个便于记忆和具有实际意义的名字。c) monitor name：MON 节点的名称，在一个集群中，每个 MON 的名称必须唯一，用于区分一个集群中不同监控节点。d) monitor map：集群中 MON 的 map，用关于 MON 间相互交流的配置文件。e) monitor keyring：集群中，MON 通信的密钥。f) administrator keyring：为 ceph 的客户端提供 admin 用户和密钥，从而使得 ceph 客户端可以和 ceph 服务器进行通信。

下面来讲述主机名为：mon1，对应 IP 地址为：192.168.100.11 的 MON 节点的部署。在 admin 服务器，通过 ssh mon1 命令，登录到 mon1 服务器上。

(2) 创建配置文件：由于 mon1 上已经安装 ceph，所以会有/etc/ceph 这个目录，在这个目录下创建并编辑 ceph.conf 文件。a) 创建 fsid，用 UUID 命令，产生一个集群的唯一标示字符串，并写到/etc/ceph/ceph.conf 文件中去，用作集群的唯一识别号。b) 添加 MON 节点主机名，编辑/etc/ceph/ceph.conf 文件，在文件 ceph.conf 的末尾处添加“mon\_initial\_numbers = mon1”行。c) 添加 MON 节点 IP 地址，编辑/etc/ceph/ceph.conf 文件，在文件 ceph.conf 的末尾处添加“mon\_host = 192.168.100.11”行。d) 创建 monitor keyring，通过 ceph-authtool 命令，创建 ceph.mon.keyring 文件，用于记录 monitor keyring 的信息。e) 创建 administrator keyring，使用命令 ceph-authtool 创建 ceph.client.keyringadministrator 文件，用于记录 administrator keyring 的信息。f) 将 client.admin 密钥导入 monitor keyring 中使用 ceph-authtool 命令将 /etc/ceph/ceph.mon.keyring 的值导入到文件 /etc/ceph/ceph.client.admin.keyring 中。g) 建立 monitor map，结合主机名，IP

地址和 `fisd` 通过 `monmaptool` 命令建立 monitor map。h) 创建 MON 节点存放目录，使用 `mkdir` 命令创建 `/var/lib/ceph/ceph-mon1` 目录。i) 初始化 MON 节点，通过 `ceph-mon` 命令，实现 `mon1` 节点的初始化。g) 配置 MON 节点对应的文件，在 `mon1` 的 `/var/lib/ceph/mon/ceph-mon1/` 目录下建立 `done` 空文件，标志着 MON 节点初始化的完成。命令：`touch /var/lib/ceph/mon/ceph-mon1/done`。在 `mon1` 的 `/var/lib/ceph/mon/ceph-mon1/` 目录下建立 `sysvinit` 空文件，标志着 MON 节点是通过 `sysvinit` 方式启动的。j) 启动 MON 节点，通过 `ceph` 命令，启动 `mon1` 节点。k) 验证 MON 节点，输入：`ceph -s`。输出结果如下：

```
cluster 86771fs2-afa3-f93b-93e0-9ce5240345a7
health HEALTH_ERR
    64 pgs stuck inactive
    64 pgs stuck unclean
    no osds
monmap e1: 1 mons at {mon1=192.168.100.11:6789/0}
    election epoch 2, quorum 0 mon1
osdmap e1: 0 osds: 0 up, 0 in
pgmap v2: 64 pgs, 1 pools, 0 bytes data, 0 objects
    0 kB used, 0 kB / 0 kB avail
    64 reating
```

此时的健康状态显示的是不正常的状态，状态中 `HEALTH_ERR`，`placement group` 为 `inactive` 和 `unclean`。但这种现象是正常现象，主要原因是 OSD 节点还没部署，统计不到对应 OSD 节点信息，等到 OSD 节点部署好后，这些状态就会发生转变。最后，设置 `ceph` 服务的启动，通过命令：`chkconfig ceph on` 实现 `ceph` 服务随服务器启动而启动。

(3) 部署 OSD 节点：通过前面部署的 MON 节点，我们可以看到在没有部署 OSD 节点的情况下，`ceph` 的状态会出现异常，只有同时将 OSD 节点和 MON 节点正常部署完成并启动后，整个系统才能正常运行起来。有一点需要注意的是，一个 OSD 节点可以部署多个 OSD Daemon 即 `osd`，`osd` 编号从 0 开始按自然数增长，接下来用主机名为 `osd1`，IP 地址为 `192.168.100.21` 的节点为例讲述下 OSD 节点的部署。其余两台服务器根据自身属性，做相应修改即可。a) 复制 MON 节点的配置文件，复制 `mon1` 服务器中的 `/etc/ceph/ceph.conf` 和 `ceph.client.admin.keyring` 到 `osd1` 服务器中的 `/etc/ceph` 目录下。复制 `mon1` 服务器的 `/var/lib/ceph/bootstrap-osd/ceph.keyring` 到 `osd1` 服务器下的

/var/lib/ceph/bootstrap-osd 目录。b) OSD 节点的准备, 通过命令 `ceph-disk`, 将数据磁盘和所对应的日志分区建立关联, 使 OSD 存储节点上的磁盘处于被激活状态。c) OSD 节点的激活, 通过 `ceph-disk` 命令将已经准备好的数据磁盘激活。d) OSD 节点分区挂载的设置, 修改 `/etc/fstab` 文件, 使得每次重启服务器时, 都能将 `sdc1`、`sdd1`、`sde1` 挂载到对应的 `ceph-X` ( $X$  为对应的 `osd` 编号) 目录下面。需要注意的是, 在 `osd1` 上, 每个磁盘挂载的对应的节点是 `ceph-0`、`ceph-1`、`ceph-2`, 在 `osd2` 和 `osd3` 上对应的目录应该是不断上增的, 而不应该再是 `ceph-0`、`ceph-1`、`ceph-2`, 根据 `osd-number` 对应的数字, 让对应的 OSD Daemon 挂载到它对应的目录上。e) 设置 `ceph` 启动属性, 通过 `chkconfig` 命令, 使得 `ceph` 服务随对应服务器的开机而启动。

(4) 扩展 MON 节点: 前面已经讲述了 `mon1` 节点的部署, 下面将讲解 `mon1` 节点的扩展。a) 复制配置文件到 `mon2` 服务器和 `mon3` 服务器上, 在 `mon1` 服务器节点上有两个配置文件 `/etc/ceph/ceph.conf` 和 `ceph.client.admin.keyring`, 将这两个文件复制到 `mon2` 服务器和 `mon3` 服务器的 `/etc/ceph` 目录下。b) 创建 `mon2` 服务器和 `mon3` 服务器对应的目录, 通过命令 `mkdir` 创建 `mon2` 服务器和 `mon3` 服务器对应的 `ceph-{mon-id}` 目录。c) 通过命令获取对应 `keyring` 值, 通过 `ceph` 命令, 分别获取 `mon2` 服务器和 `mon3` 服务器的监控器的 `keyring` 值。d) 通过命令 `ceph` 获取 `monitor map`, 通过 `monitor map` 可以实现 `mon1` 服务器和 `mon2` 服务器、`mon3` 服务器间的通信。e) 将 `mon2` 服务器和 `mon3` 服务器添加到 `monmap`, 通过在 `mon2` 服务器和 `mon3` 服务器上, 执行 `monmaptool` 命令, 将 `mon2` 服务器和 `mon3` 服务器添加到 `monmap` 中。f) 初始化 `mon2` 服务器和 `mon3` 服务器的数据目录, 通过执行 `ceph-mon` 命令, 实现对 `mon2` 服务器和 `mon3` 服务器的初始化。g) 将 `mon2` 服务器和 `mon3` 服务器添加到集群中, 通过 `ceph` 命令, 结合服务器名称和对应 IP 值, 将 `mon2` 服务器和 `mon3` 服务器添加到 `ceph` 集群中。h) `mon2` 服务器和 `mon3` 服务器的初始化, 分别在 `mon2` 和 `mon3` 服务器上, 创建 `done` 和 `sysvinit` 文件, 代表初始化对应服务器的完成和服务器的启动方式。k) 启动 `mon2` 服务器和 `mon3` 服务器, 通过 `/etc/init.d/ceph` 启动 `mon2` 服务器和 `mon3` 服务器。l) 修改 `mon2` 服务器和 `mon3` 服务器启动, 通过 `chkconfig` 命令, 实现让 `mon2` 服务器的监控服务和 `mon3` 服务器的监控服务随服务器的启动而启动。

到此，ceph 的搭建已完成，ceph 的基本功能已可以正常运行，各个服务器间可进行相互通信。其网络图如图 5-17 所示。

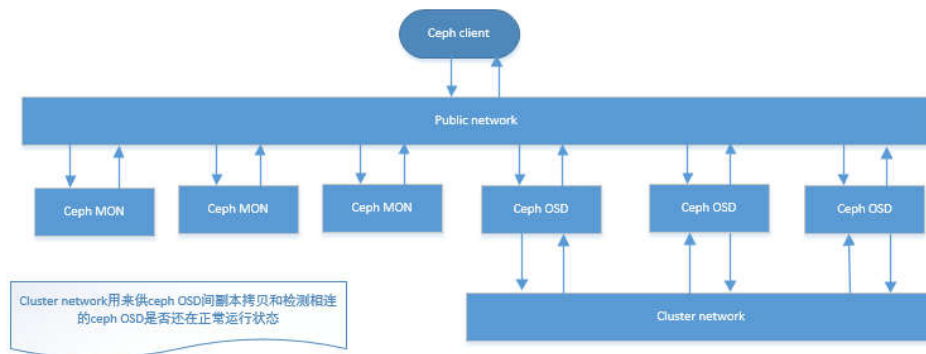


图 5-17 ceph 网络结构图

在图 5-17 中，我们能看到，有一个公共集群管理网，用于管理监控服务器和 OSD 服务器间通讯。而在 OSD 存储节点间，又有另外一个内部网络，用于实现对 primary OSD 存储节点从外界获得的存储数据进行备份，实现了利用内部网对数据的备份。

### 5.1.2 calamari 框架的搭建

在 5.1.1 节我们做好了对 ceph 的部署，在整个架构中，我们是通过调用 calamari 对应的接口，去获取所需要的信息的，接下来我们来看下 calamari 部分的部署。

(1) 克隆 calamari：通过 git 工具分别将 calamari.git、calamari-clients.git、Diamond 代码克隆到本地库。

(2) 建立 calamari server 的 rpm 包：在 calamri 文件夹下，通过可执行文件 build-rpm.sh 建立 calamari server 的 rpm 包。

(3) calamari server 的安装：通过 yum 安装 calamari-server\*\*\*.rpm（版本中 calamri server 对应的安装包）包，实现对 calamri server 的安装。

(4) calamari client 的安装：在安装 calamari client 前，需要安装 calamari client 的依赖包，通过 yum 安装 npm、ruby、rubygems、ruby-devel 工具，再通过 npm 安装 grunt、grunt-cli、bower、grunt-contrib-compass 软件，通过 gem 更

新安装包，并安装 compass。完成前期工作后，正式进入 calamari client 的安装。进入 calamari-client 文件夹，下载对应依赖包，解压 calamari-clients 对应的压缩包，再对应解压包中做相应的配置，既可以实现对 calamari client 的安装。

(5) calamari 初始化：通过 calamari-ctl 工具实现 calamari 的初始化。

(6) 在 calamari server 端配置防火墙：通过 iptables 命令实现对 4505、4506、2003、2004 端口的开放，从而使得 calamari server 能更好的进行通信。

(7) salt-minion 和 diamond 的安装：在 Diamond 的文件夹下通过 make 命令实现安装包的构建，再通过 yum 命令实现 salt-minion 和 diamond 的安装

(8) 在 ceph 节点上配置 salt-minion 并启动：修改/etc/salt/minion.d/下的 calamari.conf 文件，添加 calamari 服务器对应的 ip 地址。修改/etc/salt/下的 minion 文件，添加 calamari 服务器对应的 ip 地址。通过 service 命令，重启 salt-minion 服务，通过 service 命令启动 diamond 服务。

到此，实现了 calamari 的部署的主要流程。这里只是概述的讲述了安装 calamari 过程中需要的步骤，实际部署中会有一些对应文件去配置。

### 5.1.3 Java 框架的搭建

在项目中，所用到的 Java 部分的框架是 springMVC 框架，由于已经有很多文档讲述这部分内容了，在此，我就不再详细讲述搭建过程了，仅做简要概述搭建过程。

使用 springMVC 框架使得业务中视图、模型、控制器有了较好的解耦，方便代码的维护、升级和利用率。

在公司项目中，我们使用 maven 去管理项目中需要的 jar 包，这样只需要写入我们需要的 jar 包的名称和对应版本号，当创建 maven 项目时，系统就会自动为我们导入对应的 jar 包。下面将创建 springMVC 的核心地方说明一下。

创建 springMVC 框架时，在创建的项目中有一个 pom.xml 的配置文件，我们需要修改这个配置文件，在配置文件里，根据需要，加入 springMVC 所需要的 jar 包。我们还需要创建一个或多个 spring 的配置文件，这些文件可以实现对 bean 的管理，也可以用作其他配置。配置好文件后，在搭建的项目里，要将这些文件包含进去，使得在项目启动时，能加载这些 spring 的配置文件。

在此，简单的讲述了 springMVC 框架，至于其他需求，可以根据自己的需要，不断填充自己需要的配置信息。

#### 5.1.4 前端框架的搭建

(1) 安装 nodejs: 可以在官网: <http://www.nodejs.org/download/> 上下载和操作系统匹配的 nodejs 版本，并安装。安装 nodejs 的过程中，npm 也会被一起安装。

(2) 安装 bower: 安装 bower 需要用到上面安装 nodejs 过程中安装的 npm 工具，执行命令: `npm install -g bower` 来安装 bower。可用 bower 工具来管理 web 应用中的软件包。

(3) 安装 gulp: 通过命令: `npm install -g gulp` 实现对 gulp 的安装。用 gulp 工具, 可以启动 web 服务, 通过配置文件可以检测 js 和 jade 文档中的语法错误, 同时能实时更新 js 和 jade 文件的修改, 将修改后的文件及时更新到指定的目录下。

安装好这些工具后, 我们就可以利用这些工具, 运行我们的项目, 通过 gulp 工具可以实时的将我们在项目中的修改更新到 Tomcat 上, 结合 chrome 浏览器, 利用其中的断点跟踪工具, 在进行前端开发时, 判断是否将修改及时更新到对应的服务器上。从而极大的方面前端界面的开发。

到此, 系统中各个环节的搭建基本完成, 相互间的通信, 通过文件的配置, 可以实现各个模块间的通信。

## 5.2 系统代码的管理

因为系统是一个团队在开发, 每个小组都只是在整个项目的一个分支, 即使在一个小组里, 每个人都会分工不同, 如何保证开发者之间在共同开发一个项目时, 能做到协调工作, 这就需要有一个有效的版本管理工具去管理整个项目, 控制整个项目的开发。我所在的公司, 使用的代码版本管理工具是 git。下面结合在工作中实际使用的功能, 讲述下 git 版本控制软件, 在代码开发中的应用。

(1) git 安装：在 <https://github.com> 网站上，根据自己系统版本下载对应的 git 并安装。

(2) 建立仓库：通过命令 `git init` 命令初始化命令所在目录，我们主要是用 git 的 clone 命令：`git clone username@hostname: /path/to/repository` 将服务器端的项目克隆下来，我们还可以通过命令：`git clone /path/to/repository` 命令克隆本地仓库

(3) 工作区：在本地仓库中，有三个区。第一，你的工作区，里面是你电脑里实际存放的文件；第二，缓存区，工作区改动并保存的文件，在没有提交之前都是放在这个区域的；第三，HEAD 区，这个区是你最后要提交代码的区域。第一个区的文件可以通过 `git add <filename>` 命令，添加单个文件到缓存区，或通过 `git add *` 将所有的工作区中改动的文件添加到缓存区。通过 `git commit -m “提交信息说明”` 命令，将缓冲去的代码提价到 HEAD 区。通过这两个步骤，实现了本地代码和远程代码交互的准备。

(4) 提交本地代码：当本地项目开发到一定程度，需要将本地代码提交到远程服务器时，通过 `push` 命令，将本地仓库的 HEAD 区的代码提交到远程服务器上。具体命令为：`git push origin master`。通过这条命令，可以将本地的代码推送到远程代码库中。

(5) 分支的使用：在代码的开发过程中，分支最常被用在做模块功能开发。有时，在原有技术上，需要添加新特性，也会单独拉出一个分支，防止新添加的特性对模块分支的影响，导致迟迟不能将模块合入主线。通过使用分支，可以减少对代码 master 分支上的维护，使得每个小组只需要维护他们所创建的分支即可。由开发小组自己去维护自己小组的分支，相对于专门维护 master 分支效率高很多。当小组将对应的功能模块开发完成，再一次性将对应的分支合并到主线上，通过这个过程，分支实现了对应功能模块开发的全过程。通过 `git checkout -b “分支名称”` 命令实现分支的创建，在创建分支后，通过 `git checkout master` 可以将当前分支变成 master 分支，在 master 分支上时，可以通过 `git branch -d “分支名称”` 命令，将创建的分支删除。在本地创建的分支，在没有推送到远程时，别人是无法看到自己在本地创建的分支的，只有使用 `git push origin <branch>` 将本地分支推送到远程服务器上，才能被其他拥有开发权限的人看到你在本地创建的分支。



(6) 本地代码更新：我们在开发的工程中，别人也在开发，同时开发一个模块就会出现冲突，最好的解决办法，就是在推送本地代码前，去拉远程代码到本地，检测是否有冲突，如果有冲突，在本地解决了冲突后，才能更好的将代码推送到远程服务器。如果是一个人开发，因为不存在同时修改同一个文件情况，就不会出现刚才所说的冲突。冲突出现在有两个或两个以上的人修改同一个文件。可以使用 `git pull` 命令或者使用 `git merge <branch>` 命令实现将远程的代码自动合并到本地，不过有时会出现冲突，这时就需要手动去修改对应的冲突文件。修改后，通过命令 `git add <filename>` 命令，标记已经成功解决冲突，并将本地代码更新为最新版本。此时我们就可以顺利将自己本地的代码推送到远程服务器端代码库。

(7) 代码回滚：在项目开发过程中，由于一时的理解偏差，或者操作不当，会出现当前 HEAD 区的代码不是你想要的代码，或者你改错了东西，使得本地 HEAD 区的项目无法正常运行起来。这时，你想回到上个 HEAD 版本，你可以通过命令 `git checkout -- <filename>` 来挽回刚才的错误操作。还有一种方法，就是从服务器上获取最新版本代码，并将本地的代码指向它。通过命令 `git fetch origin` 获取最新主线代码，通过命令 `git reset --hard origin/master`，实现将本地代码指向上一个版本的代码，从而实现代码的一次回滚。

上述内容，结合了在实际工作中用到的，关于版本控制，新建分支开发新功能模块的需求等实际需求功能，讲述了在工作中 `git` 版本控制器对应的应用。

### 5.3 总结

通过本章的讲述，对在项目中的开发环境的部署有了大致的了解，前期环境的搭建使得后期的开发变得特别方便。使得整个项目结构清晰。升级维护方便，提高了代码的利用率。通过结合实际工作中的需求，讲述了分布式版本控制工具 `git` 的基本功能，运用 `git` 的基本功能，为各个开发小组提供了版本控制的基本需求。

## 第六章 系统开发和实现

### 6.1 开发工具及环境

在这个项目中，我主要负责前端运维界面的开发，着重讲述下在系统开发过程中，在前端所使用到的工具和开发环境。

(1) 软件环境：windows10

(2) 硬件换件：16G 内存 + 256G 固态硬盘

(3) 开发工具：IDEA 14.1.7 + JDK 1.8 + Tomcat 7.0 + chrome 浏览器 + Beyond compare

### 6.2 云存储运维系统的开发与实现

#### 6.2.1 前期准备

(1) 项目的克隆：使用 git 的工具，克隆开发项目对应的远程服务器端上的代码库到本地。根据各个小组开发功能的不同，创建自己小组对应的分支。在后期开发时，直接将主分支代码，切换到对应小组分支，并在自己的分支上做相应功能开发。

(2) 创建工程：a) 在 IDEA 官网：<http://www.jetbrains.com/idea/download/#section=windows> 下载 IDEA 集成开发工具，并安装。b) 打开 IDEA，点击 File —> Setting 选项，配置 IDEA 默认编码方式。在这里进行对编码的设置，是为了统一编码方式。不同开发人员，若采用不同编码方式进行编码时，当进行代码合并时，会出现乱码。同时为了防止中文乱码，公司统一采用 UTF-8 的编码方式。在此，处配置编码还有一个好处，就是每次创建新的文件时，文件的编码方式自动转化为 UTF-8 编码方式。如图 6-1 所示，将 IDE Encoding 和 Project Encoding 设置为对应的编码。

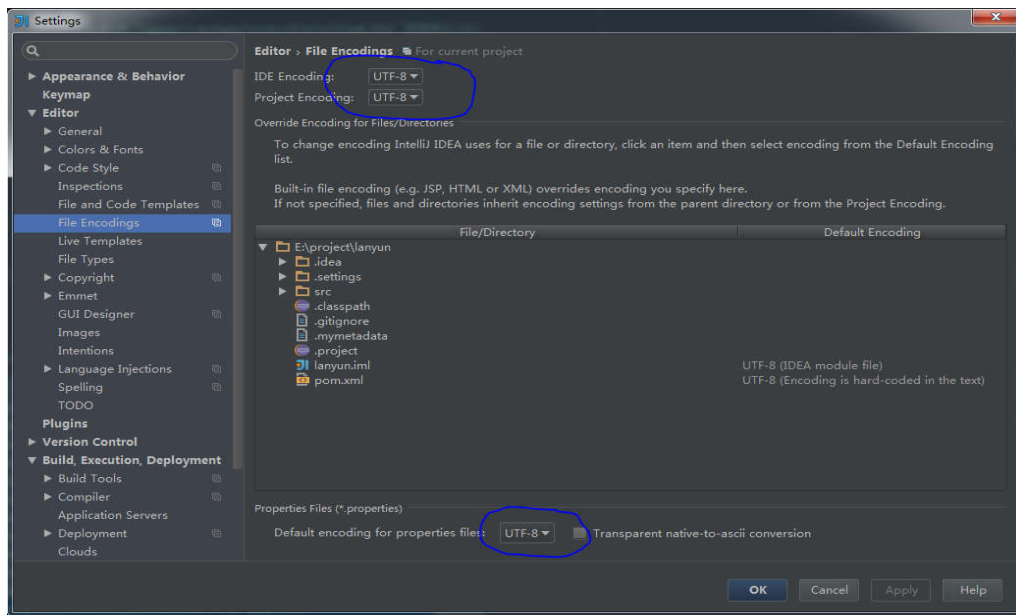


图 6-1 设置 IDEA 默认编码方式

c) 重新打开 IDEA 选择: Import Project, 导入从远程服务器克隆的项目, 如图 6-2 所示。

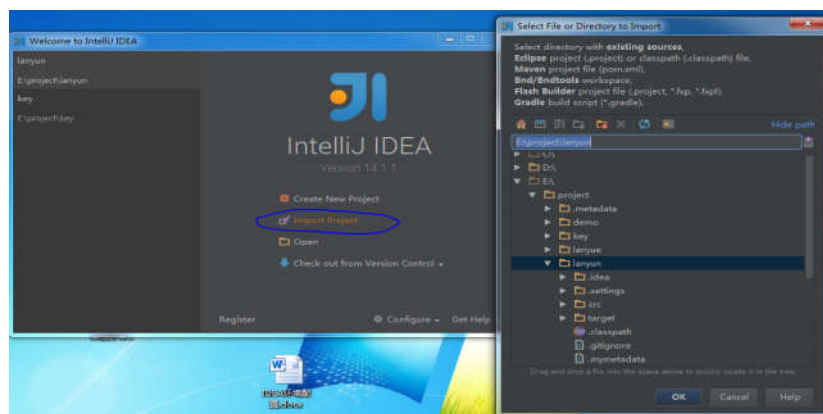


图 6-2 添加项目

d) 选中所选项目, 点击 OK, 进入相应模式选择对话框, 选择 maven 模式, 如图 6-3 所示。

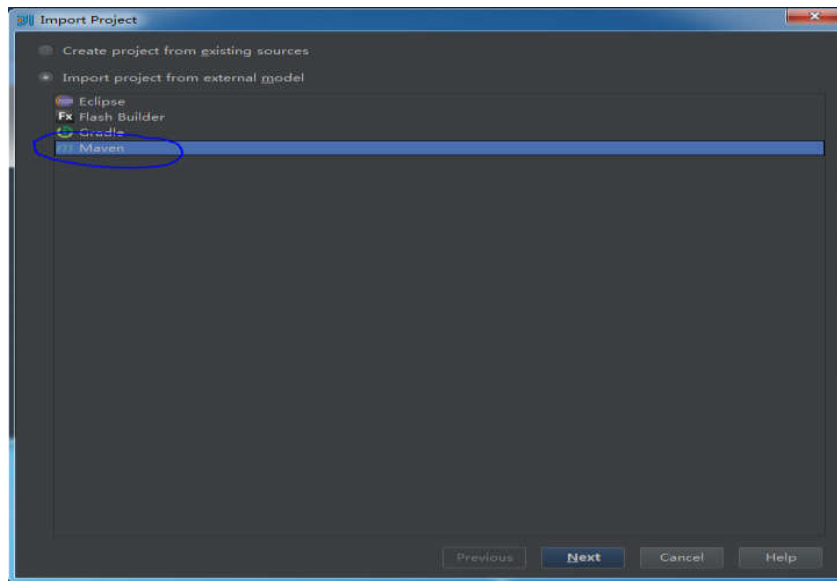


图 6-3 maven 项目选取

e) 点击 Next，直到“Please select project SDK”界面，选择 JDK 选项，并配置本地配置的 JDK，如图 6-4 所示。

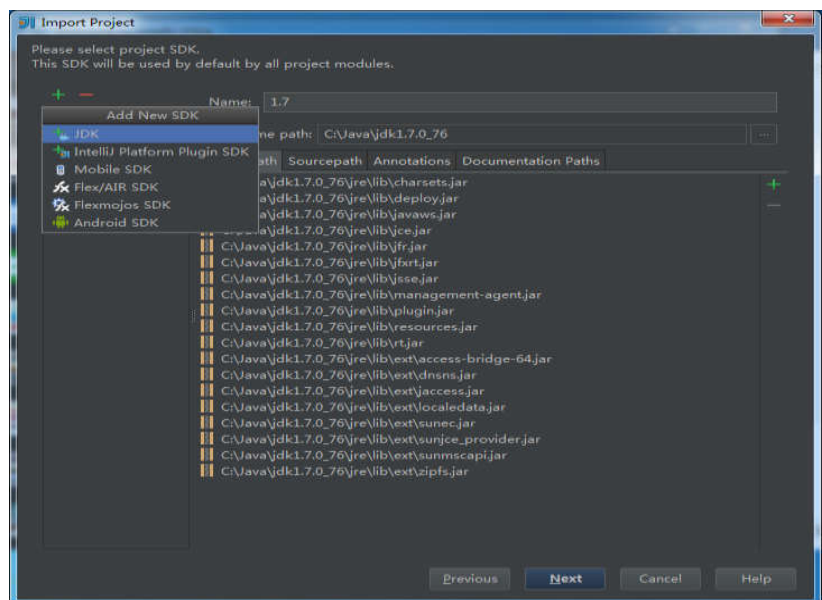


图 6-4 Please select project SDK 界面

(3) 点击 Next 直至完成。到此，项目的常见工作就已经全部完成。

(4) 运行环境的配置： a) 在 IDEA 界面，点击 File—>Setting 选项，通过下图的选项，选择 Build Execution Deployment 选项下的 Application Servers 选项，在图弹出的右边区域，点击绿色“+”图标，选择弹出框中的 Tomcat Server 选项，配置对应的 Tomcat server 信息。具体操作，如下图 6-5 所示。

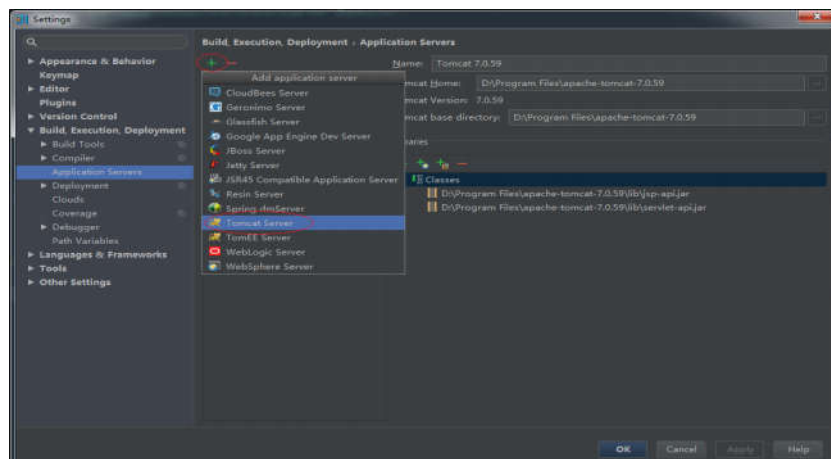


图 6-5 Tomcat Server 选项卡

b) 点击 OK，在工程项目里，点击 Run—>Edit configurations...选项，会弹出一个对话框，点击对话框左上角的绿色“+”图标，找到 Tomcat Server 选项，点击此选项会弹出二级选项卡，选择 Local 选项，图像化界面如下图 6-6 所示。

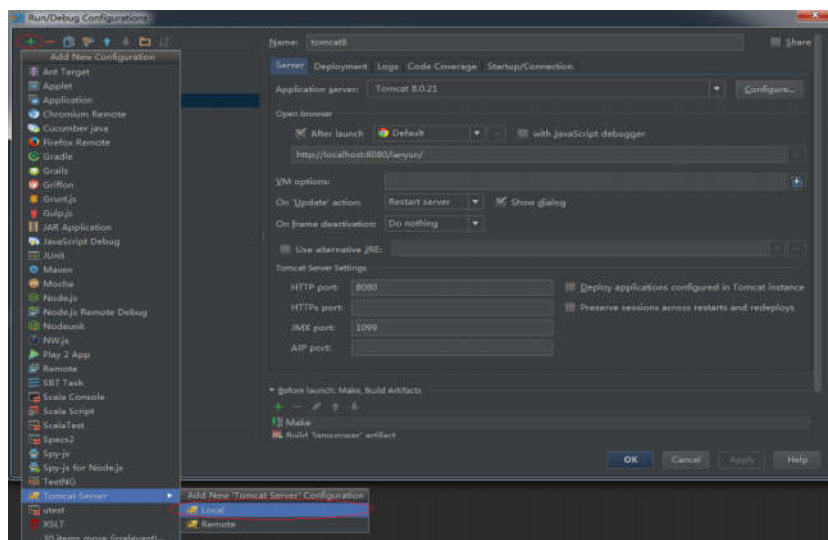


图 6-6 Tomcat Server 设置

c) 选中 Local Tomcat Server 后，设置对应的配置界面。如下图 6-7 所示。修改对应“Name”值，“Name”值用于显示在运行选项中展示的信息。没有特殊要求，可以根据项目设置成和本项目相关的一个名称。“Application Server”选项用于配置对应的 Tomcat server。

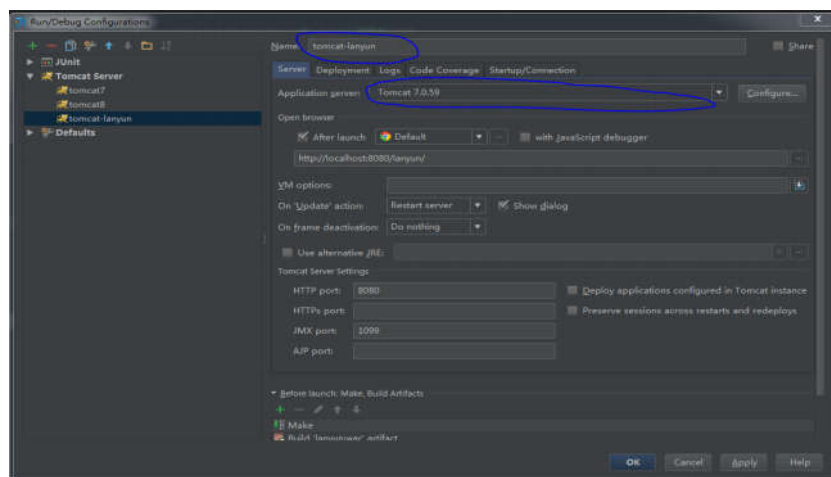


图 6-7 Server 选项的配置

d) 点击 server 选项卡旁边的 Deployment 选项卡，用来添加 Aritifact。如下图 6-8 所示，点击标记范围内的绿色“+”图标，在弹出的两个选项中选择“Artifact...”选项。

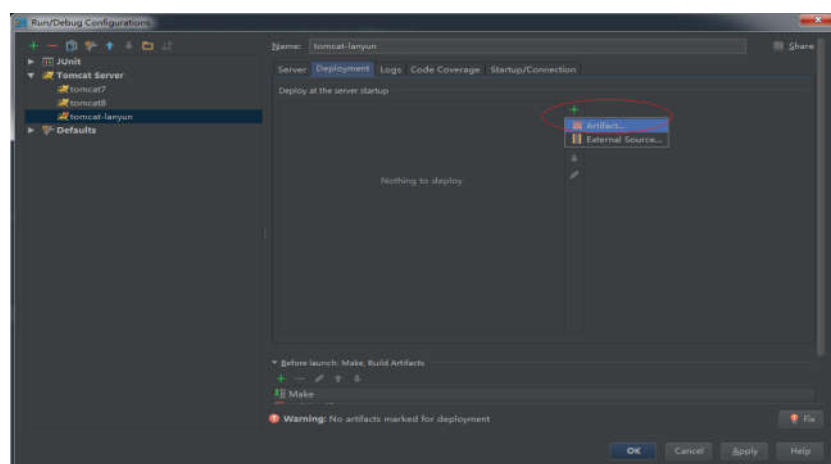


图 6-8 Deployment 选项

e) 点击“Artifact”选项，会弹出“Select Artifacts to Deploy”选项卡，在选项卡中选择如图 6-9 所示的选项。

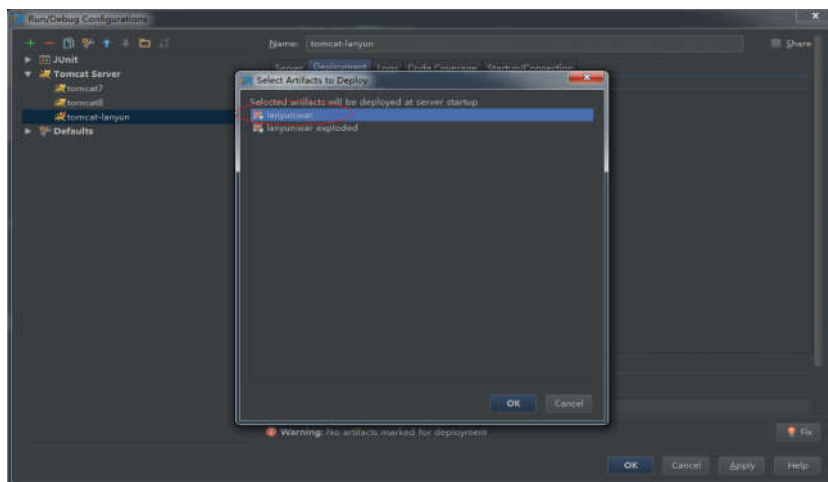


图 6-9 Artifact 选项

f) 经过上面操作后会有一个“Application context: ”对应框，根据需要，填写相应的路径，我们项目是 manage 路径，如下图 6-10 所示。

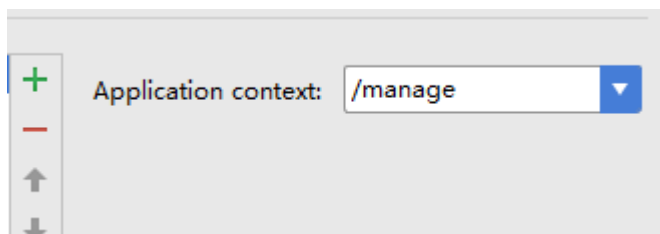


图 6-10 路径设置

到此，运行环境的配置就基本完成了。进入 IDEA 项目中，根据实际需求进行项目的开发。

### 6.2.2 运维系统的实现

(1) 前端界面的开发：当用户进入运维管理界面，用户要能获取所有集群列表信息，可以通过每个集群的“详情”连接，可以查看对应的每个集群的信息

概览，在众多信息概览里，要突出主要信息，即磁盘容量和磁盘使用量的信息。通过详情，用户可以查看每个 cluster 类中所包含的详细信息。具体活动图如图 6-11。

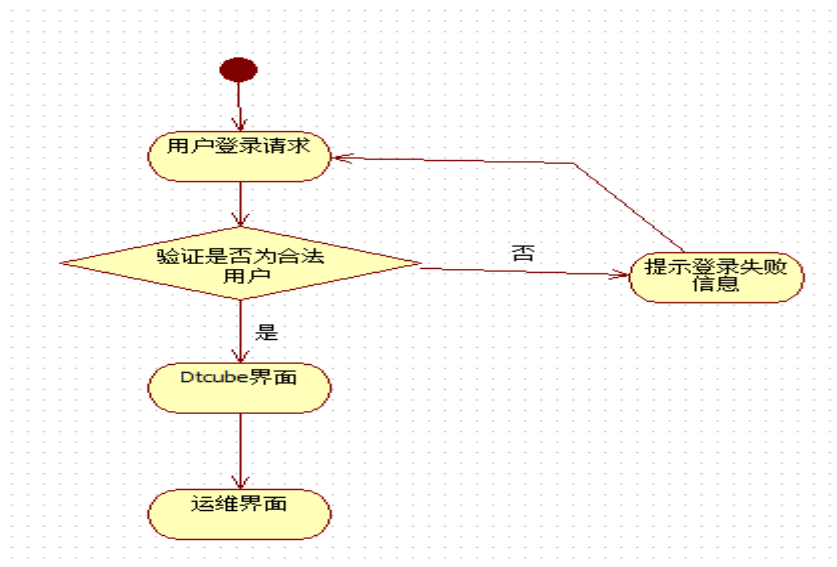


图 6-11 用户活动图

在 cluster 集群的首界面，会罗列所有的集群，在集群列表中展现集群最重要的信息，即存储信息，如下图 6-12 所示，显现了集群列表信息。



6-12 存储概览



此处的环境只有一个集群，如果有多个集群会在这个集群下方罗列出来。通过“详情”选项，我们可以查看，此集群的详细信息，进入集群详细界面，如下图所示 6-13 所示。

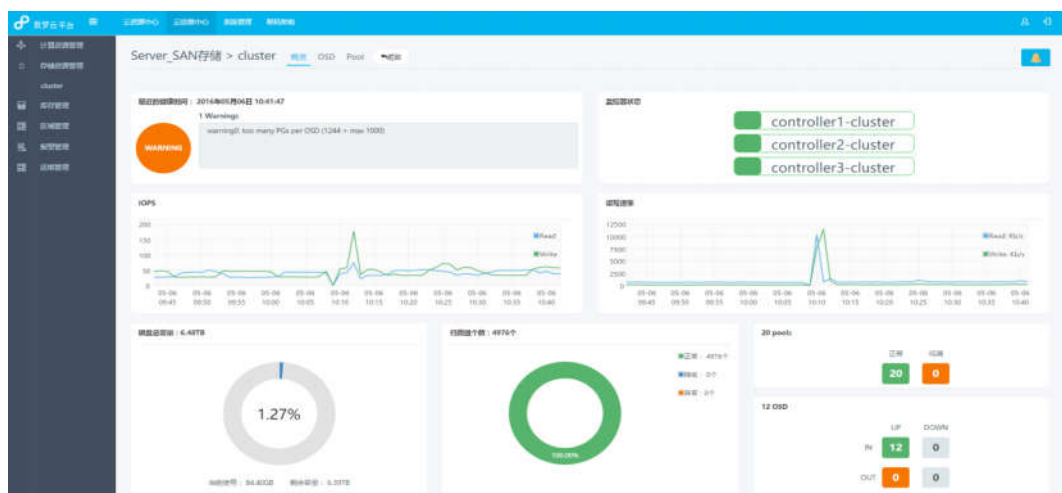


图 6-13 集群信息概览

通过图 6-13，我们可以更加直观的感受下集群中所包含的一些信息。针对其中比较复杂的两个模块 OSD 和 Pool 又单独的开发出来两个选项，用于详细展现 OSD 和 Pool 里面的详细信息，在 OSD 界面，展现了在每个机架上所对应的服务器和部署在服务器上面的 OSD 信息。如图 6-14 是 OSD 界面。

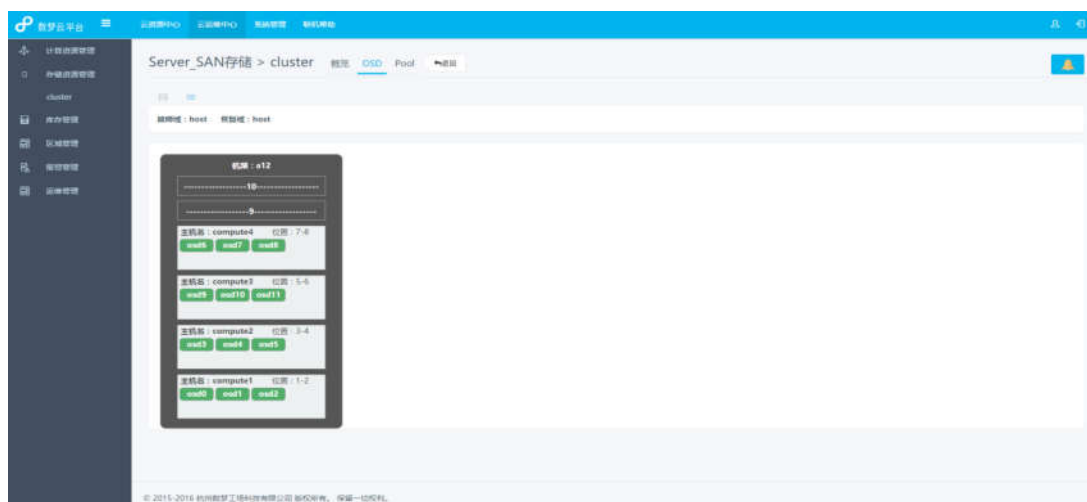
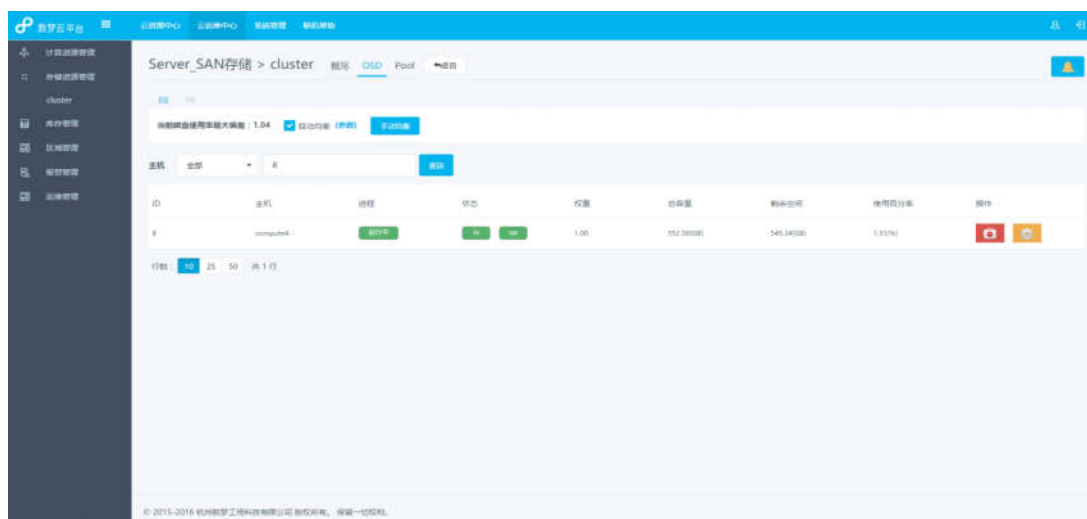


图 6-14 OSD 在机架上的布局

通过图 6-14 我们可以看到，这个集群所有 OSD 存储节点所在的机架位置和对应的服务器。我们可以通过点击故障域上面的小图标，展现所有 OSD 的详细列表，我们还可以点击对应 OSD 查看每个 OSD 的详细信息，如图 6-15 是通过点击机架上 OSD8 所得到的关于 OSD8 所对应的信息表。

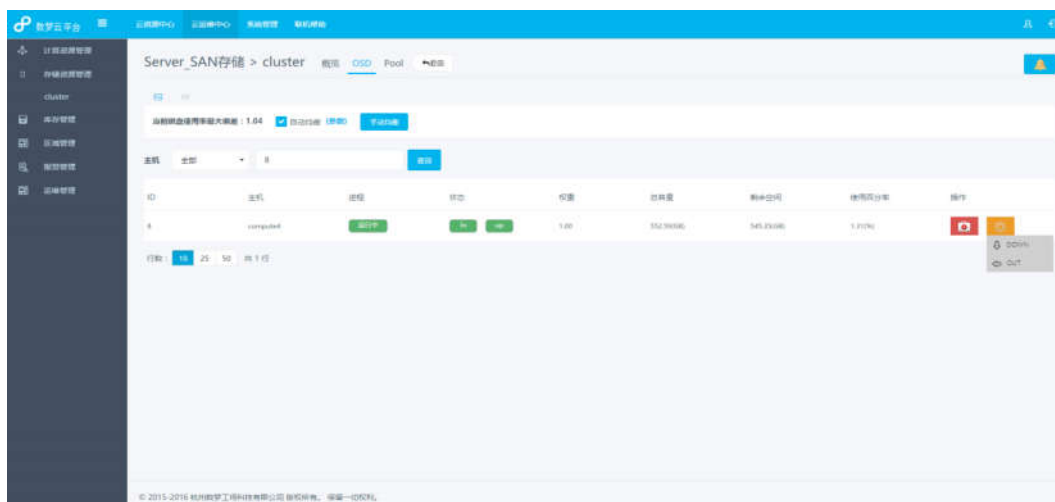


The screenshot shows the 'Server\_SAN存储 > cluster' page with the 'OSD' tab selected. It displays a table of OSDs with columns: ID, 主机 (Host), 进程 (Process), 状态 (Status), 权重 (Weight), 总容量 (Total Capacity), 剩余容量 (Remaining Capacity), 使用百分比 (Usage Percentage), and 操作 (Actions). The first row shows an OSD with ID 8, host 'compared', and a status of '运行中' (Running). The table has 16 rows in total, with the first row highlighted.

ID	主机	进程	状态	权重	总容量	剩余容量	使用百分比	操作
8	compared	运行中	OK	1.00	552.58GB	545.84GB	9.91%	[Stop] [Refresh]

图 6-15 OSD 详细列表

在这里我们可以看到每个 OSD 所对应的详细信息，同时我们可以通过点击后面的按钮进行 OSD 状态的设定，具体情形如图 6-16 所示。



The screenshot shows the same OSD detailed list as Figure 6-15, but with the 'down' button highlighted in the '操作' (Actions) column for OSD 8. The button is labeled 'down' and has a red 'x' icon.

ID	主机	进程	状态	权重	总容量	剩余容量	使用百分比	操作
8	compared	运行中	OK	1.00	552.58GB	545.84GB	9.91%	[down] [Refresh]

图 6-16 OSD 设置

左边的按钮对应的信息是副本校验和深度校验两个选项，在 OSD 详细信息界面，我们可以看到有对应的均衡选项。手动均衡界面如图 6-17 所示。

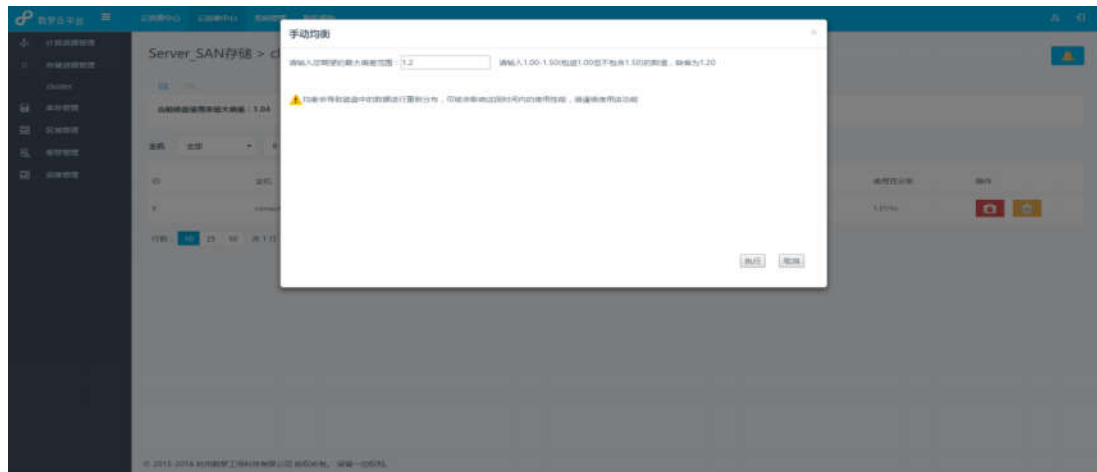


图 6-17 手动均衡界面

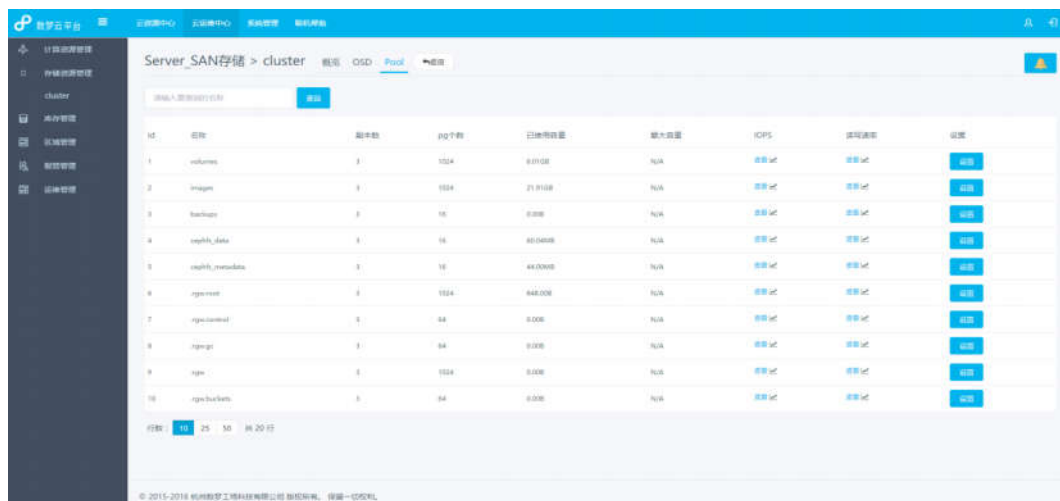
在手动均衡界面，我们填写相应的数值，点击执行，即可立刻进行均衡操作。因为均衡操作对用户存储操作影响较大，所有会在均衡操作下面给出一段警告提示，告知使用者，手动均衡带来的后果。

手动均衡旁边有个自动均衡，当长时间使用存储空间时，就会出现数据在磁盘分布不均的情况，当有磁盘损坏或有新的磁盘插入时，也会出现磁盘分布不均衡，通过自动均衡选项，重新分配数据分布，自动均衡界面如 6-18 所示。



图 6-18 自动均衡界面

在存储池界面记录着 Pool 的详细信息，我们可以看到一个集群中所有 Pool 的列表信息，以及每个 Pool 所拥有的副本，所具有的 pg 个数，磁盘总容量和已经使用过的磁盘大小。详细信息如图 6-19 所示。



The screenshot shows the 'Server\_SAN存储 > cluster' interface with the 'Pool' tab selected. It displays a table of storage pools with columns for ID, Name, Replicas, PG Count, Used Space, Total Space, IOPS, and Write Speed. Each row has a 'View' button.

ID	名称	副本数	pg个数	已使用容量	最大容量	IOPS	读写速率	设置
1	volumes	3	1024	8.01GB	N/A	查看	查看	编辑
2	images	3	1024	21.91GB	N/A	查看	查看	编辑
3	backups	3	16	8.08GB	N/A	查看	查看	编辑
4	cephfs_data	3	16	40.04GB	N/A	查看	查看	编辑
5	cephfs_metadata	3	16	44.00GB	N/A	查看	查看	编辑
6	rgw-root	3	1024	448.0GB	N/A	查看	查看	编辑
7	rgw-control	3	64	0.00GB	N/A	查看	查看	编辑
8	rgw-obj	3	64	0.00GB	N/A	查看	查看	编辑
9	rgw	3	1024	0.00GB	N/A	查看	查看	编辑
10	rgw-buckets	3	64	0.00GB	N/A	查看	查看	编辑

图 6-19 Pool 的详细列表

在详细列表中有 IOPS 属性，用于表示 Pool 磁盘中，当进行数据读、写时所对应的速率值，通过点击“查看”对应的连接，我们可以查看到每个 Pool 所对应的 IOPS 在近一段时间内的值的变化。如图 6-20 所示。

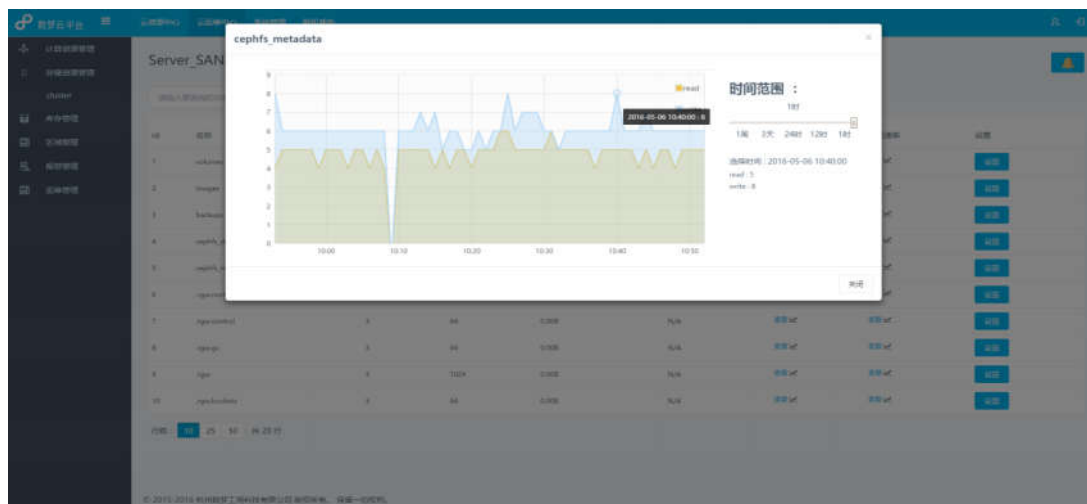


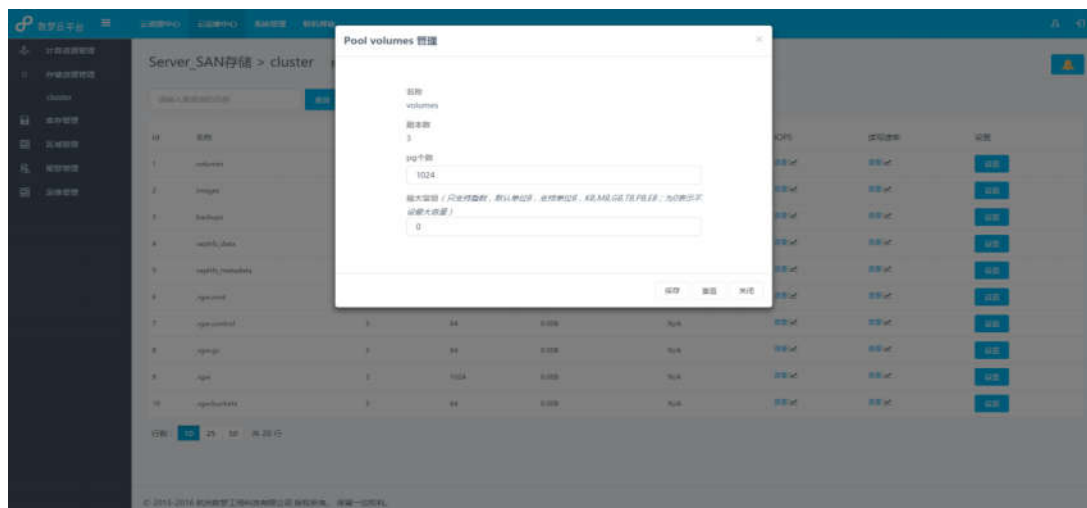
图 6-20 cephfs\_metadata 的 IOPS 的信息

通过 6-20，我们可以看到 IOPS 对应的信息，其中单位用次每秒。当我们点击 Pool 列表里读写速率的查看时，我们可以看到对应 Pool 的读写速率的记录。读写速度的信息如图 6-21 所示。



6-21 cephfs\_data 的读写速率信息

我们还可以通过点击每个 Pool 记录后面的设置，进行对应的 Pool 的设置。可供设置的属性包括 pg 个数和磁盘最大容量。通过图 6-22 可以将这部分的信息展现出来。



6-22 Pool volumes 的设置

到此，通过图形化界面展现了云存储运维管理系统的前端界面，前端界面的所实现的功能几乎就这么多，随着后期的需求，会在相应的界面添加对应的功能模块。

(2) 后台功能的实现：在原本的框架中，前端到后台 Java 交互的是利用的 angularjs 中的 \$resource 模块，初始化一个变量 v，然后用 v 调用 \$resource 模块中的 get( function ( res ) )，最终将结果返回给 res 参数。后面因为云存储运维界面的特殊需求，我们开发了，适合我们云存储运维系统的调用模块。我们通过重新编写后台调用模块，实现适合我们自己界面的后台交互模块。我们通过 JavaScript 中的 Promise 模块，在 Promis 中使用 XMLHttpRequest 机制，实现了更灵活的处理异步函数的调用模块。

前台 angularjs 控制器传过来的 url 请求，会走到使用 springMVC 搭建框架中的控制器模块，即通过 @Controller 标记的类。系统会找到对应的 Java 代码中，用作控制器模块的 Java 代码，再通过 @RequestMapping 层层匹配，找到 angularjs 发过来的请求所对应的 Java 代码块。到此，实现了前台 angularjs 到后台 Java 代码的调用。

在控制器端的代码块接收到来自 angularjs 发过来的请求后，通过封装，将请求发给中间用于做数据处理的模块，即 service 模块。这个模块用于转发请求，同时处理从更底层返回的数据，实现对数据的拆分、组合，从中提取用户需要的信息，去掉用户不感兴趣的信息，并将数据封装成 json 格式的字符串（方便前端解析、识别、处理），再负责将数据转发给上一层。这一层会将上一层发送过来的请求，进行本层的相应封装，然后继续向下转发给和 calamari 部分交互的 Java 代码层，我们称为 calamariApi 层。

在 calamariApi 层，会接受来自上层 service 层发过来的请求，通过进一步的封装，调用 CloseableHttpClient 类的 execute() 方法，再通过配置文件，实现和底层 calamari 的交互，再有 calamari 调用自身的功能模块实现，封装函数，实现对 ceph 模块的调用。因当时主要负责前端和 Java 代码的开发，对于 calamari 的实现机制和底层调用不了解，在此就不去叙述这部分的代码实现机制。

到此，已将我在项目中，在云存储运维系统开发的过程中所涉及到的，关于软件开发的模块，大部分都已罗列、讲述出来。这些是关于 Server SAN 模块的开发，还有关于 IP SAN、FC SAN 的开发，其实现机理大致相同，在此就不在累述。

### 6.3 总结

在本章开始阶段，讲述了项目的开发环境和使用的工具，以及工具的使用，开发环境的搭建，这样对整个项目的开发有了更全面的了解。在本章中间一节讲述了项目中所涉及到的功能模块，展示了用户界面，同时也更详细的讲述了底层实现的过程。

## 第七章 总结

感觉时间犹如白驹过隙，瞬间到了毕业的时间，在实习的过程中有很多坎坷，阻碍，成就和喜悦。回顾这一年来的实习经历，确实感觉自己在逐渐成长，并有相应的收获。接下来，讲述下我在实习的过程中遇到的，困扰我好久的几个问题。

### （1）数据请求延时

#### a) 问题描述

在云存储运维系统模块中，部分界面在从后台取数据时，有时需要很长的时间。数据获取时间的长短和后台维护的磁盘容量成正比，当磁盘容量很大时，数据获取的时间会比较长。当用户点击界面时，就会发送相应的请求到后台，取相应的数据。如果要读取的信息还没有返回，用户就切换到其他界面。而在另外的界面就又会发送这个界面相应的请求，而在这个界面又没返回时，又做界面切换，在连续快速的做页面切换时，就会造成后面再去做页面跳转时，即使用正常的速度切换界面，在页面数据信息显示时，有时用时也会是正常情况下的好几倍。在操作较为复杂的界面时，还会出现页面卡顿，有时还会出现卡死状态。

#### b) 现象分析

首先需要说明的一点，JavaScript 语言的引擎是单线程的，不支持多线程操作，支持异步操作，并强制所有的异步事件排队等待执行。对界面加载缓慢这个问题，经过断点跟踪，最终发现，当采用项目框架的方法，用\$resource 模块中的 `get( function ( res ) )` 往后发送请求时，项目中对应的 js 文件就会采用异步的方式等待后台返回结果。在发送请求的 js 代码块等待返回结果的同时，js 文件会处理后面其他程序，在结果返回前去做页面切换，发出请求数据的 js 文件将会继续停留在内存等待数据的返回。其实，在切换界面的那一刻，前面界面对应的 js 文件对请求返回的数据，就已经没有实际意义了。如果快速在多个界面做页面切换，就会有很多无效的 js 文件继续停留在内存，等待发送的请求返回数据。当对应的数据返回后，有相应接收函数还要去做数据返回后的相应处理。这样当对多个界面进行快速界面切换时，就会出现在同一个时间段，有大



量的前台 js 文件并发做相应处理，而这些处理中，只有一个是有用的 js 线程。如果其他 js 线程抢占了处理器的处理权，那么其他所有等待处理的 js 线程只能等待当前的 js 线程处理完自己的任务。这时对用户有用的 js 线程就只能等占用处理器的 js 线程释放了处理器的使用权后，再去和其他无用的 js 线程抢占处理器使用权。通过这个过程，我们可以了解到，为什么在快速连续切换界面时，界面数据信息显示会变得特别慢。

当知道这些时，对卡顿的理解也变得比较容易了。当有用的 js 文件获得处理器的使用权时，去处理数据的过程中，当遇到异步操作时，处理器的使用权会立刻被其他 js 线程抢占。所以当前 js 线程即使需要一个非常短暂的异步等待，这也会使得当前 js 线程失去处理器使用权，从而使得有用 js 线程，要和其他线程，去争夺释放了的处理器的使用权，这也是在界面上，为什么会出现卡顿的原因。

还有一个现象，就是当快速对页面进行切换时会出现界面卡死。就是不论等多久，界面的数据始终无法正常显示。显然这个现象不仅仅是卡顿这么简单。刚开始对这个现象确实感到困惑，同时，这个现象还不容易复现。以至于这个现象使我困惑了好久。后来通过不断的追踪调试，发现，在固定几个界面切换时，这个现象较容易复现。我就试着在这几个界面，不断通过断点调试，终于找到了问题出现的原因。做快速多界面切换时，在当前界面被加载时，有其他界面对应的 js 文件处理无用的数据，这时有可能会和当前界面对应的属性标签进行交互。如果恰巧无用的 js 线程在做数据处理时，调用了现在界面对应的标签，做数据处理，就有极大地可能出现数据需求格式异常，造成数据处理异常中断，界面就会停止加载。出现前面所说的卡死状态。到此，就可以完全知道问题描述中出现问题的根源所在。

### c) 解决方法

当知道具体造成这些问题的原因时，就要想法去解决这些问题。通过查看，尝试项目中所有向后台发送请求的模块，没有一个可以完美解决上述现象的模块。我就希望能通过新的数据传输方法框架，去实现运维界面需求的数据的调用。通过不断查阅资料，发现有 Promise 机制，可以很好地解决当前所遇到的问题。

Promise 有三种状态：pending（默认），fulfilled（完成），rejected（失败）。其中 pending 可以想其他两种状态进行转变。解决上述问题的方案：通过调用

XMLHttpRequest 机制，向后台发送数据。之所以使用 XMLHttpRequest 机制，是因为这个机制可以很好的控制数据请求状态，在 XMLHttpRequest 模块中，可以通过调用 XMLHttpRequest 模块中的 send 函数实现对请求的发送，通过 XMLHttpRequest 中的 onabort 函数可以实现对请求的终止。而这个特点，可以完美的解决我们在界面中出现的问题。通过 Promise 模块对这个获取数据请求的机制进行封装，实现 angularjs 框架中需要的异步的特性。可以很好的实现数据获取的异步特性。

在 angularjs 框架下，当点击页面跳转时，js 线程会捕获到 \$destroy 事件，通过在 \$destroy 事件中添加对应的函数调用，实现在界面跳转时做相应的工作，以实现页面跳转所需要完成的事件。利用在 XMLHttpRequest 模块中，具有立刻终止数据请求的函数的特性，结合 Promise 实现机制，可以在界面发生跳转前，利用 \$destroy 函数，使得发送数据请求的代码块，可以在界面跳转前得到相应的回应。这样在没有进入下一个界面前，就可以处理完所有需要等到请求返回后的处理事件。这个处理过程也不会出现 js 线程竞争处理器资源的现象，即不存在异步事件排队等待处理这个现象。在跳转到新的界面时，上个界面对应的 js 线程，处理结束后就退出内存的占用，不会影响到新界面对应的 js 线程的运行。从而从根本上解决了上面提到的关于界面反应迟钝、卡顿和卡死的现象。后面会在附录中列出发送请求机制。在附录中会附加上这部分实现机制。

## （2）浏览器间兼容性

### a) 问题描述

所有功能模块开发时，使用的都是 chrome 浏览器。在基本模块开发完毕之后，在做浏览器兼容性测试时，发现运行在 IE 浏览器下，页面显示异常，云存储运维管理系统上面的数据完全显示不了，而项目中其他模块的数据可以正常显示。

### b) 现象分析

通过断点定位发现，点击、刷新或切换页面时，在后台 Java 代码部分，没有收到任何关于前端代码发过来的请求。由此可以断定，问题出现在前端向后台发送数据部分。通过单步跟踪，发现在自己编写的往后台发送请求的模块，在使用 IE 浏览器时出现了问题。这也是为什么别人开发的功能模块可以正常在 IE 浏览器下运行，而我们模块却出现了页面无法正常加载，数据无法正常显示的情况，后在网上查阅，发现在 IE 浏览器上不支持 Promise 异步机制。到这，

算是找到了问题所在。下一步就是如何解决在 IE 浏览器上，关于 Promise 的异步特性的兼容性问题。

### c) 解决方法

起初，将解决问题的重点放在了找到一种代替 Promise 异步机制的功能模块，实现相应的异步请求操作。后来发现很难找到能很好的做到解决页面加载慢、卡顿和卡死的异步机制。在思索中，受到 echarts 插件的启发，心想按照 Promise 实现机制，手动写个模块，用于支持相应的异步操作，在搜寻关于 Promise 实现机制的时候，发现有单独的 Promise 模块 promise-0.1.1.min.js 可供下载使用。我就下载了对应的 Promise 模块，并加载到了工程项目里。这样，在调用 Promise 模块时，不在依赖浏览器是否支持 Promise 模块，从而使得项目在不同浏览器上，在做向后台发送数据时，能更好的运行。

通过这个浏览器兼容的问题，我意识到，框架里对应的功能实现模块，大都是经过了架构师精细推敲而搭建的，如果某些特殊需求需要自己的实现框架，就一定要兼顾很多方面，比如浏览器的兼容性，自己加载的模块是否和原有模块有冲突，自己模块的性能，客户是否能接受等等。

接下来讲述下我在开发中犯过的入门级的错误。当时的错误也是由于不熟悉版本控制器 git 所造成的。项目组分给了自己一些任务：开发相应模块。当自己将相应模块开发完毕时，需要将自己开发的代码合入小组分支。正常情况下，首先将自己的添加的文件，通过 git 中的 add 命令，添加到本地缓存区，再通过 git 的 commit 命令，将本地添加和修改的文件提交到本地代码库中。此时需要和远程的对应分支进行交互。通过 git 中的 pull 命令，将远程的代码拉倒本地库，此时，如果有冲突会报相应冲突，并需要借助 Beyond compare 比对工具，手动去解决冲突。在解决完后需要重新执行 git 中的 add 命令和 commit 命令，以表示已经解决相应冲突。如果不报冲突，即可实现远程代码和本地代码自动合并，自己本地代码是最新最全的代码。在执行 git 中的 push 操作，即可实现本地代码和入到小组开发分支。

由于对代码何如经验不足，当遇到需要将自己的工作合入对应分支时，我当时的做法是：克隆一份代码，切换到我们项目组开发的分支，通过 Beyond compare 比对工具，将自己开发的和远程最新分支进行比对，将新添加的部分更新到我的本地开发代码库。如果开发的功能复杂点，如要一周时间完成，这样，再通过比对工具，将远程分支新添加的代码更新到自己本地库，将是非常

费时和费力的事。当时在这个问题上，就出现过严重的问题，我花费了一天的时间去将我开发的代码合入远程我们开发的分支。这种手动合入，还经常会出现文件冲突的现象。就是当自己更新代码时，别人也在更新代码。因为我这种手动合入代码的方式耗时太长，所以大大增加了代码冲突的概率。当将远程的代码合入到本地时，在执行 git 中的 `add` 和 `commit` 命令，将本地代码提交到本地代码库中，在利用 git 中的 `push` 命令，实现代码的提交。

由于每次合入代码，都是手动检查异同。当时总是感觉，代码合入远程分支，事件非常复杂和耗时的一件事。后来通过和同事聊天，得知他们将自己的代码合入主线，正常情况下，只需要短短的几分钟就可以实现代码合入。通过咨询才知道正确的合并代码的方法。通过这个事件，使我知道很多时候，不是所做工作操作复杂，而是自己没有很好的掌握其中的技巧。同时这也展现了工作中和同事交流的重要性。

在实习的过程中，曾一度进入一个困惑：当面临新语言、新技术时，如何去面对。不光在实习，就是以后的工作中，也总会遇到新的技术或新的语言，当我们遇到这些新的事物时，我们该如何去面对这些事物？下面说下我在我的实习过程中所经历的感悟。

实习期间，有段时间做界面开发，其中就用到了 HTML5、JavaScript、CSS 等语言，当时的感觉是既然在用这些语言，就应该尽快将这些语言中所有的知识都去掌握。于是就去买对应的书籍，准备去系统的学习。在学习的过程中，又会出现新的事物，在看这些书籍的过程中有用到了 `nodejs`、`openstack` 等相关的知识，根据刚开始的思想，毫无疑问，又去买对应的书籍去系统的学习。刚开始还好，涉及的东西不是太多，实习工作不大，但后期又有很多新的知识出现。当时的感觉，时间完全不够用，毕竟每天还要去做小组分给自己的任务。看书只能抽时间。因为看书是通读，又因为时间紧，有好多东西根本不能很快消化掉，这样时间久了，以前看的又会有好多都忘记了，感觉这种方式对待新出现的事物，效率特别低。同时，看到的好多知识，在实际开发中几乎从来未曾出现过。因为将自己好多时间用在通读书籍上，所以感觉在实际工作中因投入的时间、精力较少，而效率较低。当时我就怀疑，我这种面对新事物的方法是否可取。通过以前的学习经验，给我的感觉是，既然遇到新知识了，就应该系统的，认真的去将这门语言所涉及到的东西认真去看一遍。在有工作的情况下，我感觉这种方法对我来说不适应。毕竟时间没有在校期间那么充裕，同时，

通读因为没有侧重点，使得吸收效率差，还有可能会遇到一些需要有一定工作经验的人去研读的东西，这些东西很容易使初始读者对新的技术产生恐惧，排斥，对新的事物失去研究的兴趣。后来通过和别人交流，结合自己的实际情况，感觉遇到新事物，应从实际需要出发，有针对性的去学习当时需要的知识模块。尽可能多的先看和自己实际工作有关的知识，如果时间够的话，再以当前学的为基点，向外不断延伸。从而实现有针对性，有明确目的的去学习新的事物，这样就可以利用有限的时间，尽可能快的掌握一种新技术。当掌握一定程度后，感觉新技术需要自己有个全面的掌握，那时再去系统的学习。这种情况下，有一定实战经验，再加上有针对性，学习效率自然而然的就会提高上来。

在整个实习的过程中，有辛酸，有坎坷也有喜悦。在实习的过程中，我感知到了基础知识的重要。通过自己的努力，使我更加坚信，只要你去努力付出，总会获得应有的汇报。这才是事业的起点，以后的路还很长，只有脚踏实地的去走，才能获得理想的效果。

## 参考文献

- [1] 赵铁柱.分布式文件系统性能建模及应用研究[D].广州:华南理工大学,2011.
- [2] Riedel Erik,Faloutsos Christos,Gibson G A.et al.Active disks for large-scale data processing[J]. IEEE Computer,2001,34(6):68-74.
- [3] Wei S A. Ceph: Reliable, scalable, and high-performance distributed storage[D].UNIVERSITY OF CALIFORNIA,2007.
- [4] Mesnier,Mike R.Ganger,and Erik Riedel.Object-based storage[J].Communications Magazine,IEEE,2003,41(8):84-90.
- [5] 龚高晟,通用分布式文件系统的研究与改进[D].广州:华南理工大学,2010.
- [6] 李翔.Ceph 分布式文件系统的研究及性能测试[D].西安:西安电子科技大学,2014.
- [7] 许敏,分布式文件系统容错机制的研究与实现[D].西安:西安电子科技大学,2012.
- [8] Weil S A,Brandt S A,Miller E L,et al.Ceph:A scalable,high-performance distributed file system [C] Berlin:Proceddings of the 7 th Symposium on Operating Systems Design and Implementation(OSDI),2006:307-320.
- [9] 廖彬,于炯,孙华等.等基于云存储结构重配置的分布式存储系统节能算法[J].计算机研究与发展,2013:3-18.
- [10] 沈良好,吴庆波,杨沙洲.基于 ceph 的分布式处处节能技术研究[J].先进计算与数据处理,2015,41(8):13-17.
- [11] 王敬轩,分布式文件系统存储效率优化研究[D].华中科技大学,2013.
- [12] 廖舒恬.安全对象分布式文件系统的设计与实现[D].华中科技大学,2013.
- [13] 段剑弓.存储系统 NAS 和 SAN 的差异和统一[J].计算机应用研究,2004,21(12):94-97.
- [14] 郑传建.Ceph 对象文件系统添加任务迁移特性的研究[D].武汉理工大学,2014.
- [15] Holmquist L E,Redstrom J,Ljungstrnd P.Token-based access to digital information proceeding [M].Berlin:SpringerVerlag,2000.
- [16] 杨飞,朱志祥,梁小江.基于 ceph 对象存储的云网盘设计与实现[J].图像·编码与软件,2015,28(10):96-99.
- [17] 蔡官明.开放式云存储服务平台设计及移动云盘应用开发[D].广州:华南理工大学,2013
- [18] Chang F,Dean J,GheMawat S, et al. Bigtable: A distributed storage system for structured data[J].ACM Transations on Computer Systems (TOCS),2008,26(2):4.
- [19] 聂瑞华,张科伦,梁军.一种改进的云存储系统容错机制[J].计算机应用研究,2013,30(12):3724-3728.
- [20] Corbett J C,Dean J,Epstein M,et al.Spanner:Google's globally-distributed database[C].Proceedings of OSDI. 2012,1
- [21] Dean J,Ghemawat S.MapReduce:simplified data processing on large cluster[J].Communications of the ACM,2008,51(1):107-113.

- [22] Zeng,Ling-Fang,Dan,and Ling-jun Qin.SOSS:smart object-based storage system[J].Machine Learning and Cybernetics,2004:26-29.
- [23] Wut,LeeW,Lin Y, etal. Dynamic load balancing mechanism base done loudstorage [C]. Computing, Communication sand Application conference(Com-Com Ap). HongKong, IEEE, 2012: 102-106.
- [24] 王芳,陈亮.对象存储系统中基于负载均衡的设备选择算法[J].华中科技大学学报:自然科学版,2007,35(10):46-49.
- [25] 符永康.云存储中数据安全关键技术研究及系统实现[D].北京:北京邮电大学,2013.
- [26] 张敏.基于对象存储文件系统研究[D].成都:电子科技大学,2012.
- [27] Yan CR, Shen JY, Peng QK, et al. A throughput-driven scheduling algorithm of differentiated service for web cluster.Wuhan University Journal of Natural Sciences, 2006, 11(1):88-92.
- [28] 谢雨来,冯丹,王芳.主动存储技术及其在对象存储中的实现[J].中国计算机学会通讯,2008,4(11):27-33.
- [29] 李青山,魏彬.Ceph 分布式文件系统的研究性能测试[J].西安电子科技大学,2014, 29(5):1-15.
- [30] Hsiao H C,Chung H Y,shen H, etal.Load rebalancing for distributed file system,IEEE Transactionson,2013,24(5):951-962.
- [31] 袁艳丽.存储系统主动队形实现机制研究[D].武汉:华中科技大学,2011.
- [32] Hyeran Lim,Vikram Kapoor,Chirag Wighe.Active Disk File System:A Distributed,Scalable File System[J].In:Proceedings of the Eighteenth IEEE Symposium,Mass Storage Systems and Technologies,2001,32(8):101-114.
- [33] 方圆.基于对象存储元数据管理策略的研究与实现[D].郑州:将防局信息工程大学,2012.
- [34] Sacks,David .Demystifying Storage Networking DAS,SAN,NAS,NAS Gateways,Fibre Channel, and iSCSI[J].IBM Storage Networking,2001,23(7):3-11.
- [35] Mesier,Mike,Gregory R.Ganger,and Erik Riedel.Object-based storage[J].Communications Magazine,IEEE,2003,41(8):84-90.

## 附 录

附件 A

Promise 异步机制的实现

```
/**
```

```
 * Created by sun on 1/11 0011.
```

```
 */
```

```
App.factory('PageHandle', ['$q',
```

```
    function ($q) {
```

```
        'use strict';
```

```
        var pageHandle = {};
```

```
        var requestMap = {};
```

```
        pageHandle.getData = function (url) {
```

```
            var req = new XMLHttpRequest();
```

```
            var promise = new Promise(function (resolve, reject) {
```

```
                req.open('GET', url);
```

```
                req.send();
```

```
                req.onload = function () {
```

```
                    if (req.status === 200) {
```

```
                        resolve(JSON.parse(req.responseText));
```

```
                    } else {
```

```
                        resolve('request error');
```

```
                    }
```

```
                };
```

```
                req.onerror = function () {
```

```
                    resolve('request error');
```

```
                };
```

```
                req.onabort = function () {
```

```
                    reject('abort handle');
```



```

    };
    req.ontimeout = function () {
        resolve('timeout')
    };
    req.onreadystatechange = function () {
        if (req.readyState === XMLHttpRequest.DONE) {
            delete requestMap[url];
        }
    }
    });
    requestMap[url] = {
        promise: promise,
        request: req
    };
    return promise;
};
pageHandle.abort = function () {
    for (var url in requestMap) {
        var req = requestMap[url].request;
        if (req.readyState !== XMLHttpRequest.UNSENT &&
req.readyState !== XMLHttpRequest.DONE) {
            req.abort();
        }
    }
};
return pageHandle;
}]);

```

## 致 谢

在论文完成之际，我要感谢我的校内导师和校外导师，对这篇文章的指导和帮助。在两位导师的悉心指导和帮助下，我才能够较顺利的完成。两位导师不仅在教学中拥有严谨的态度，并且平时在思想上和生活中都给予了我极大的启发和帮助。在查找参考文献的过程中导师也为我提供了大量有价值的资料。再次由衷的感谢两位导师的付出和照顾。感谢我的母校，在校期间，学校不仅提供了良好的学习环境，还为我的知识储备和完善专业知识提供了很多的帮助。感谢母校对我的培养，给与我展现自我的机会。也正因为母校，才使我在追求知识的道路上，认识了身边这么多优秀的朋友和同学。他们是我快乐的源泉，他们在我不断的探讨和研究的过程中，给与我很多帮助，陪我一起成长。

在我一年的实习生活中，单位的领导给与我极大的帮助。对于工作和生活中遇到的问题，领导都会耐心而热情的为我解答，给与问候。并且对于此次论文研究的课题，领导给与我很多建议和启示。同时，单位提供的良好的工作环境，使我在工作之余，有能力和时间来完善自己的论文。感谢单位给我这次实践的机会，这次机会将为我今后的工作奠定基石。

我需要感谢在背后默默付出，不求回报的父母。在 20 多个春秋岁月里，父母为我遮风挡雨。用一生中最美好的光阴，支撑了一个避风港湾，才使我能够茁壮的成长，并且受到良好的教育，感谢父母在学习和工作中对于我的支持和帮助。

时光荏苒、近三年的研究生生活马上结束，即将离开母校，使我的心情久久不能平复。这里是我梦开始的地方，也是我挥洒汗水的起点。这近三年的研究生生活，使我更好的磨练了我的意志、完善了我的性格，使我在生活中拥有良好的心态，使我能更好的去适应社会、贡献社会，成为国家的栋梁人才。赠人玫瑰，手留余香。再次衷心感谢老师、领导和同事对我这次论文的指导和帮助。