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： 软件系统设计

导师姓名： 周英华

完成时间： 二○一六年八月三日

University of Science and Technology of China

**Limited**

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： The design of software system

Supervisor： Ms. Yinghua Zhou

Finished time: August 3nd, 2016

书脊

|  |
| --- |
| **基于C**  **E**  **P**  **H的云存储运维系统的设计和实现**  **孙君伟**  **中国科学技术 大学** |

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。从而实现具有备份、容灾、易于维护特性的存储系统。基于ceph的云存储系统充分利用每一个磁盘，能提供高稳定性和高并发的特性。同时具备易于扩展的特点。

基于ceph的云存储运维系统是基于ceph基础之上对磁盘进行监控的系统，在ceph层可以实现的功能如下：

1. 容灾备份：

当出现突发情况时，造成局部磁盘数据损坏而造成的数据丢失，可以通过ceph的容备特性，等排出故障磁盘后通过存储在其他地方的数据来恢复原先坏掉磁盘的数据。对于一些重要的信息，以实现将原来的两地三中心，变成三地三中心，值要不同时破坏掉对应数据的所有存储场所，对应的数据都是安全可靠的

1. 高效存取：

现在存储数据PB级数据已经很常见了，EB级数据也时有出现，这就面临着一个问题，如何更好的存储这些数据。对于大数据，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的云储存运维系统，提供了很好的存储方案。其设计充分发挥了每台对应服务器的计算能力，摆脱了传统单一中心节点的瓶颈，实现了真正无中心的特性。其中有八个字概括这一特性：无需查表，算算就好。这充分体现了整过数据存取的过程中不会因单点故障，而造成系统故障，同时保证了存取的高效

1. 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，以及不同磁盘集聚的磁盘使用率的失衡，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。避免单个磁盘因频繁被操作而缩短使用寿命。

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ABSTRACT

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 cloud storage system based on CEPH makes full use of every disk, which can provide high stability and high concurrency. At the same time, the cloud storage system has the characteristics of easy to expand.

CEPH, which based cloud storage operation and maintenance system, is the monitor system, based on the disk. In the CEPH layer, we achieved the function as follows:

1. back and up:

It is possible which in the emergency situation, the disk in someplace was damage and resulting in the disk data was loss. Using the CEPH which has the characteristic of content preparation, the disk data can be recover when replacing the broken disk by other place disk data. For some important information, it has three places storage data and it have three center point of primary, rather than two places for storage data and three center point of primary. Using the system, if not make the same data was destroy at the same time, the corresponding data are safe and reliable

1. efficient access:

Now the data which stored disk has TB level is very common. And the data which has PB level was also appearance. That will make a problem, how to efficient store these data. For big data, effectively store is essential. It is not only related to the safety of data storage, but also includes the efficiency of reading data. The cloud storage system based on CEPH provides a good method for storing data. The design gives full play to the computing power of each corresponding server, gets rid of the bottleneck of the traditional single center node, and realizes the real non center characteristic. There are eight words summed up the characteristics: no look-up table , it is of for calculating. This fully reflects that the whole process of data access will not cause the system failure as a single point breakdown, while ensuring access to efficient

1. 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. To avoid a single disk has shorten the service life due to frequent operation.

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, concurrent

1. 绪论
   1. 选题背景和意义

（1）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。

（2）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[1]。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。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。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，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，现在产生的数据是PB级数据。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，将会很好的满足这种需求。

* 1. 国内外研究发展状况及发展趋势

1.2.1 ceph概述

此次毕业论文设计，是基于ceph[2]分布式系统的，接下来让我们看下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功能[3]。从而使其满足不同应用需求下的部署和运维。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。

1. IO路径太长：

这种情况在客户端和服务器端都有存在，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。

1. 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。

1.2.2 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，可靠的性能[4]。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：

优势：基于ceph的云存储运维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。且在对设备进行升级或扩容时，给客户提供极大便利，基本不会影响客户正常业务，节省了客户维护方面的开销[5]。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[6]。从而保证了数据的安全，并且不影响客户的正常操作。随着供应商的日益增多，使用费用将会不断降低。

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[7]。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2x1011千瓦时，约占全球总电量的1.3%，且其比例还在不断增加[8]。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%[9]。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，负载均衡体现的效果不是很好[10]。

1.2.3 基于ceph的云存储运维系统发展趋势

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[11]。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。

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[12]。

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. 云存储系统设计的先进性和意义

对于个人：当多人协作，共同完成一件事时，就会涉及到信息的一致性。采用网络数据共享可以很好的做到信息同步，并能将自己的工作及时告知他人，防止出现重复劳作，同时保证了其他人可以及时获得自己工作的内容以便他们后续工作。

对于企业：通过基础ceph的云存储系统，可以节省一大笔开销，比如在更新设备，升级服务，存储管理等方面，基于ceph的云存储系统显然比企业独立存储系统占优势[13]。同时通过基于ceph的云存储系统，在磁盘故障方面有很好的保证，基于ceph的云存储系统会自动迁移故障磁盘数据，故障恢复快，使企业不必担心因磁盘故障造成的业务中断[14]。在磁盘扩容，服务升级方面有着孤立磁盘无法比拟的优势[15]。

* 1. 本人主要工作

当前负责运维系统的前端界面开发和用户前端与服务器交互的后端开发。还做磁盘性能测试。

1.4 论文组织

1. 理论概述

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。随着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。此时，云储存就自然而然的应运而生。

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。而云存储具备：

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，政府他们将来预测不到自己将来需要多少存储。如果还是采取传统方式，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。而如果才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，可以申请扩容，方便便捷。避免了闲置浪费和后期因存储紧张造成的不便。

1. 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。从而极大的提高了存取效率。

1. 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，极大的发挥系统的并发性能。让客户在为大文件的存储而感到烦恼。

1. 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，有些追求私密，有的追求完整，有的追求高效率。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，云储存还能够对对应的数据进行加密保护。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，才用这样存储，防止非法用户通过单个存储节点获取所有对应信息。

2.2 ceph机制

2.2.1 集群框架机制

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，进项相应的操作。

1. 元数据服务器

用户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

1. 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

1. 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，或者当系统主插入了先的存储设备，集群监控器能感知这些变化，并作出相应信息的更新，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，使的基于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。如图2.2是ceph存储机制：



图2.2 ceph存储机制

通过图2.2我们可以看到，存取的实现主要有以下几个部分：

1. file -> object

实现方式是给每个文件一个唯一id，然后再将较大的文件拆分成事先规定好的，易于在网络中传输的各个块，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，从而将文件拆分成易于存储的objects，方便后续操作。

1. objects -> 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 -> 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。

1. PGs -> OSD

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，可以得到一组中有n个的osd，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 -> object -> PG -> 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，从而实现高并发特性。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，也是为磁盘高效率做出了很大的贡献。随着不断向网络磁盘存储数据，网络磁盘数据就会出现分布不均衡的现象。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OSD的状态，OSD的状态包括：

up&in：该OSD运行正常，且至少承载了一个PG。

up&out：该OSD运行正常，但并未承载任何PG。

down&in：该OSD发生异常，但仍然承载至少一个PG。

down&out：该OSD发生异常，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。OSD会根据cluster map进行数据维护，而client则是根据OSD进行数据的寻址。从而使得基于ceph的云储存具有高效的存取性能。

2.2.3 高可用性机制

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很多小的数据单元被分别存储在不同的OSD节点上，这样，同一个文件的不同OSD间会有通信。当ceph云存储系统中有磁盘出现故障，那么和故障磁盘上文件组成同一个文件的其他OSD节点都会检测到这个磁盘的故障，当其他OSD磁盘检测到这以故障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。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。这些过程可以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，更重要的是其消除了单点瓶颈问题。

第三章 需求分析

3.1 系统概述

1. 前端：

合法用户通过验证后，可进入运维中心，从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，可以选择性的进入对应存储池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和Server SAN。下面分别看下他们对应的详细信息。

FC SAN：在FC SAN中包含有POOL的信息，记录着对应FC SAN所具备的存储池的个数，每个POOL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定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，下面将分别叙述Cluster下对应的信息：

1. Cluster健康状态：

这个信息展现了cluster的最近更新的健康时间点，其中包含健康、警告、失败三种状态。当出现警告或失败时，有对应的警告或失败的对应信息。

1. 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。

1. IOPS：

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。

1. 读写速率：

记录了最近一段时间内cluster的读写次数的变化情况。单位为MB每秒。

1. 磁盘总容量：

记录了对应cluster总的磁盘大小以及使用情况。

1. 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，没有达到实现规定的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

1. OSD：

OSD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，每隔OSD节点记录了每个OSD的ID编号，OSD所属的主机，OSD状态，OSD容量信息，OSD已用空间信息以及每个OSD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的状态。

OSD是整个云储存的核心，OSD的存储状态直接影响到整个存储系统的性能，其中说道的负载均衡，就是指在OSD上的数据是否处于良好的存储状态。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，决定是否进行自动均衡。为了达到很好的用户体验，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，让系统自动做均衡处理。这样可以做到不影响用户正常操作的情况下，调整OSD存储的数据，来达到负载均衡。

1. 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的数据和读写速率数据。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本POOL最大的空间容量。

1. 后端：

此处所说的后台包括后面的Java代码，和Java交互的Calamari部分，和Calamari对接的ceph环境，以及和ceph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。

1. 系统构造

通过（1）、（2）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的实例将对应的实例，通过结构图将系统的整体结构展现出来。下面将分段将系统结构图

有系统进入Server SAN的结构图如图3.1所示：

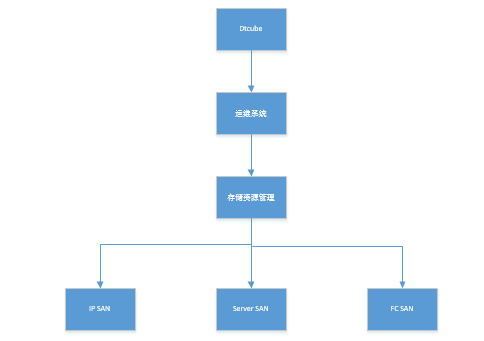


图3.1 系统结构图

下面图3.2是在Server SAN结构内部对应的结构图：

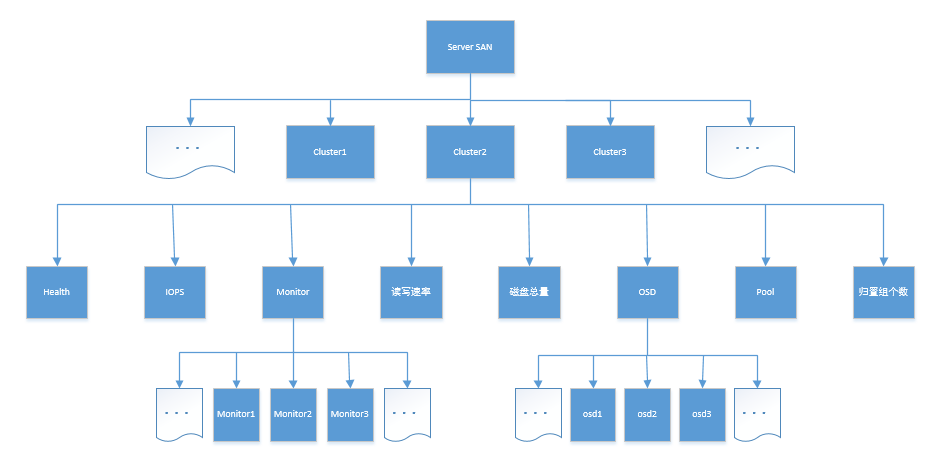


图3.2 Server SAN内部结构图

3.2 系统总体架构

此处说下基于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存储系统智商对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，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，极大的简化了设计繁琐。Angular JS有时前端用户界面的控制器，可以很好，方便的实现和用户界面的互动，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3.1所示：

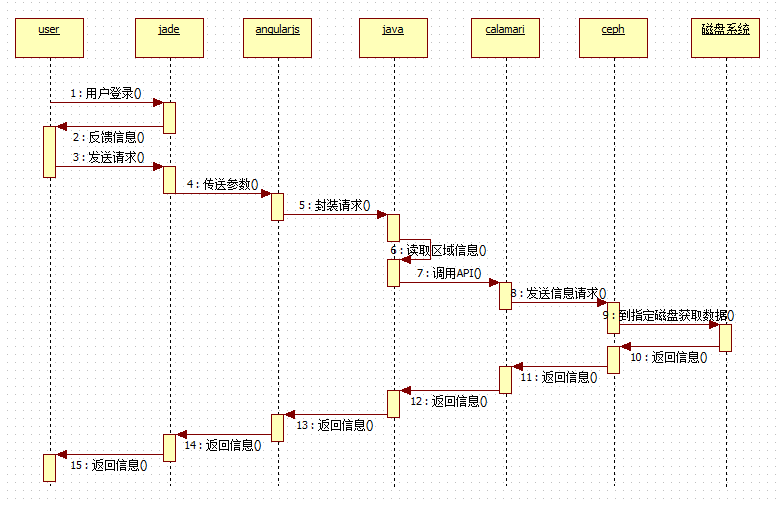


图3.1 业务请求的总体流程

3.3 云存储运维系统性能需求分析

随着数据信息的急剧增加，公司，企业，政府等部门对不同信息的不同要求，他们有个各式各样的数据存储需求，将他们对存储主要性能需求如下：

* + 1. 高性能需求：

Ceph存储机制实现了所有数据的存储和读取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ceph存储机制我们可以知道，一个文件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同时工作，实现了高并发的特性。

用于存储数据的每个OSD节点都有对应的权重值，通过权重值可以实现负载均衡，从而更充分发挥每个OSD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重要的方式就是创建副本。在基于ceph的存储系统中，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通过内网创建对应的副本，这样就可以避免有client去创建副本而造成的网络消耗。

* + 1. 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，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。同时这种实现机制还可以防止脑裂。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存储节点就会感知到故障OSD节点，就会激发ceph存储系统自动恢复机制，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，极大地提高了用户体验。

* + 1. 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节点，使得每个存储节点具备更多的处理能力，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，自动均衡机制会执行自动均衡操作，实现数据的均匀分布，不在需要人工去干预。更容易的实现固件的删除、增添和固件的升级。

* 1. 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。当出现磁盘数据出现分布极端化时会影响数据的存取操作，使得系统的整体性能有很大的下降。所以系统给提供了均衡机制，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。自动均衡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的操作的影响。手动均衡操作会使得系统立刻做均衡操作，应慎重使用。

* 1. 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的让每个层次之间尽可能的松耦合，以便后期随着用户需求的不断增加和系统升级的需要，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的便利。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。

1. 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，用于管理项目中所需程序包

1. 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。

1. 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一些工具，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，在执行相应命令就会自动加载到指定位置，为以后的扩展提供极大便利。

4.1.2 后端架构设计

后端包括好几个部分，通过这几个不认识的整个项目在解耦方面做得比较好。讨论下后端代码的实现。

首先是和前台交互的是Java代码，在Java代码里使用了springmvc框架，对前面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ceph改进的REST API接口。当calamari接收道上层传输的请求后，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，从而实现对相应数据信息的请求。

4.2 云存储底层设计

通过讲述一个超小型的ceph云存储系统集群来了解下云存储底层的大致设计理念。试过在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是，不会影响到外界信息的传输。

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服务器是的磁盘，在OSD服务器上有5块磁盘，其中一个做系统盘，按安装时分配了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，剩下有添加三块存储空间较大且相等的磁盘，用作存储数据。同时将添加用作日志的磁盘，将空间划分为三部分，分别用来记录每台OSD服务器上的磁盘操纵信息。

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个，防止出现一方事故，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。通过这种设计可以测试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、工具、云存储底层的一些基本情况，下面章节将更加具体的介绍系统的设计和实现，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 存储底层基础环境安装和部署

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，此处所讲述的框架，更倾向于独立讲述各个模块所涉及到的东西。在整个项目开始前，需要将项目所运行的换将搭建起来，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存储系统给的搭建

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：

此处只做演示ceph集群搭建的展示，而非之际工程中的配置方案。在本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。位置为：centos 7.2， 1个CPU，1GB RAM，只需一个20GB的磁盘做系统盘就可以，磁盘分区格式为XFS。对于OSD服务器，需要用三个OSD Daemon用于存储条带化的数据，需要三个OSD Daemon用于做数据的备份。这种需在可以通过安装三台OSD服务器，其中每台OSD服务器上安装三个OSD Eaemon，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下表所示：

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，其中admin、mon1、mon2、mon3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，记录相对应数据磁盘的操作，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。

1. 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。通过下面命令实现更新服务器时间，同时开启ntp服务：

ntpdate 0.cn.pool.ntp.org

hwclock –w

systemctl enable ntpd.service

systemctl start ntpd.service

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：

systemctl disable firewalld

systemctl disable firewalld

为了更好高的通信，还需禁用SElinux：

sed -i s'/SELINUX=enforcing/SELINUX=disabled'/g /etc/sysconfig/selinux

为了防止在安卓集群的各个节点是不报错，需要注销掉sudoer中的requiretty选项：

sed -i s'/Defaults requiretty/#Defaults requiretty'/g /etc/sudoers

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}，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。

1. 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：

1. 系统盘：保持不动，安装时自动做了相应分区和格式化。
2. 日志盘：将日志盘分为三个分区，给个分区占整个磁盘的33%。格式无特殊要求。
3. 数据盘：每个磁盘用作一个分区，且将磁盘格式化为xfs。

下面将对日志盘和数据盘进行分区和格式化，具体格式化过程如为：

日志盘：

1. # parted / dev / sdb
2. ( parted ) mklabel gpt
3. ( parted ) mkpart pirmariy 0% 33%
4. ( parted ) mkpart pirmariy 34% 66%
5. ( parted ) mkpart pirmariy 67% 100%
6. ( parted ) quit

执行完上诉步骤，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。

数据盘：

1. # parted / dev / sdc
2. ( parted ) mklabel gpt
3. ( parted ) mkpart pirmariy 0% 100%
4. ( parted ) quit
5. # mkfs.xfs / dev / sdc

上述是对一个OSD服务器的一个一个数据盘进行分区和格式化的，其他的磁盘也做相似操作。将其中过得sdc换成sdd和sde。通过上面操作，完成了对所有OSD服务器的所有数据磁盘的分区和和格式化。

1. 安装ceph：
2. 在安装ceph之前需要导入key：

rpm --import 'https : // ceph.com / git / ? p = ceph.git ; a = blob\_plain ; f = keys / release.asc'

1. 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。

1. 安装yum插件yum-plugin-priorities：

yum -y install yum-plugin-priorities

1. 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。

1. 清除缓存并更新源：

执行下面命令：

yum clean all

yum update

通过生面命令可以清除原有的yum缓存，并更新最新定义的yum源。

1. 安装第三方软件包

执行下面命令：

yum -y install snappy leveldb gdisk python-argparse gperftools-libs

通过这个这条命令可以将ceph依赖的软件包都安装到服务器上，从而使得能够正常安装ceph。

1. 安装ceph

yum -y install ceph

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：

部署MON节点前需要去了解一些基本概念，后面的部署会用到这些概念性的东西。

1. fsid：每一个集群拥有一个fsid，且每个集群的fsid唯一。
2. cluster name：集群名字。
3. monitor name：MON节点的名称，在一个集群中，每个MON的名称必须唯一。
4. monitor map：集群中MON的map，用关于MON间相互交流的配置文件。
5. monitor keyring：集群中，MON通信的秘钥。
6. administrator keyring：为ceph的客户端提供admin用户和秘钥，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登录到mon1服务器上。

1. 创建配置文件：

由于mon1上已经安装ceph，所以会有/ etc / ceph目录，在这个目录下创建并编辑ceph.conf文件

1. 创建fsid：

用UUID产生一个集群的唯一标示字符串，并写到/ etc /ceph / ceph.conf中去

1. 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

1. 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

1. 创建monitor keyring：

ceph-authtool --create-keyring /etc/ceph/ceph.mon.keyring --gen-key –n mon. –cap mon 'allow \*'

1. 常见adminstrator keyring：

ceph-authtool --create-keyring /etc/ceph/ceph.client.admin.keyring --gen-key -n client.admin --set-uid=0 --cap mon 'allow \*' --cap osd 'allow \*' --cap mds 'allow'

1. 将client.admin密钥导入monitor keyring中：

ceph-authtool /etc/ceph/ceph.mon.keyring --import-keyring /etc/ceph/ceph.client.admin.keyring。

1. 建立monitor map：

利用主机名，IP地址和fisd建立mornitor map，通过下面命令实现：

monmaptool --create --add {hostname} {ip-address} --fsid {uuid} monmap

1. 创建MON节点存放目录：

mkdir /var/lib/ceph/mon/ceph-mon1

1. 初始化MON节点：

ceph-mon --mkfs -i mon1 --monmap /etc/ceph/monmap --keyring /etc/ceph/ceph.mon.keyring

通过上述步骤基本完成了对MON节点的部署，还需要完成下ceph.conf文件，最终文件如下所以：

[global]

fsid = 86771fs2-afa3-f93b-93e0-9ce5240345a7

mon\_initial\_members = mon1

mon\_host = 192.168.100.11

auth\_cluster\_required = cephx

auth\_service\_required = cephx

auth\_client\_required = cephx

filestore\_xattr\_use\_omap = true

public\_network = 192.168.100.0/24 # public network

cluster\_network = 192.168.101.0/24 # the private network for OSD copy data

osd\_pool\_default\_size = 3 # set the times of object

osd\_pool\_default\_min\_size = 2 # write twice is allow in degraded state

osd\_pool\_default\_pg\_num = 512

osd\_pool\_default\_pgp\_num = 512

osd\_crush\_chooseleaf\_type = 1

1. MON节点初始化：

在mon1相应目录下建立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相应目录下建立sysvinit空文件，标志着MON节点是通过sysvinit方式启动的。

1. 启动MON节点：

/etc/init.d/ceph start mon.mon1

1. 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 creating

此时的健康状态是HEALTH\_ERR,placement group 为 inactive和unclean主要原因是因为OSD节点还没部署，等到OSD节点部署好后，这些状态就会发生转变。

1. 设置ceph服务随服务器一起启动：

chkconfig ceph on

1. 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的状态会出现很多异常，只有同时部署了OSD节点和MON节点，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，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。其余两个节点根据自身属性，做相应修改即可。

1. 复制MON节点的配置文件：

复制MON节点/etc/ceph/ceph.conf、ceph.client.admin.keyring到 OSD主机的/etc/ceph目录。

复制MON主机/var/lib/ceph/bootstrap-osd/ceph.keyring到 OSD主机的/var/lib/ceph/bootstrap-osd目录。

1. OSD节点的准备：

通过准备节点将磁盘和所对应的日志分区建立关联，执行 命令如下：

ceph-disk prepare /dev/sdc1 /dev/sdb1

ceph-disk prepare /dev/sdd1 /dev/sdb2

ceph-disk prepare /dev/sde1 /dev/sdb3

1. OSD节点的激活：

通过命令将已经准备好的磁盘激活，命令如下：

ceph-disk activate /dev/sdc1

ceph-disk activate /dev/sdd1

ceph-disk activate /dev/sde1

1. OSD节点分区挂载的设置：

修改/ etc / fstab文件，是的每次重启服务器时，都能按预先设定的目录去挂载OSD节点里的数据盘。执行命令如下：

vim /etc/fstab

在fstab文件中添加下面的几行：

/dev/sdc1 /var/lib/ceph/osd/ceph-0 xfs defaults 0 0

/dev/sdd1 /var/lib/ceph/osd/ceph-1 xfs defaults 0 0

/dev/sde1 /var/lib/ceph/osd/ceph-2 xfs defaults 0 0

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了，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。

1. 让ceph系统随开机时启动：

chkconfig ceph on

下次服务器重新启动时，ceph服务器会一起启动

1. 扩展MON节点：

前面已经讲述了mon1节点的部署，下面将讲解MON节点的扩展。

1. 复制配置文件到mon2服务器和mon3服务器上：

在mon1节点服务器上有两个配置文件/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。

1. 创建mon2服务器和mon3服务器对应的目录：

mkdir /var/lib/ceph/mon/ceph-{ mon-id }

1. 通过命令获取对应keyring值：

ceph auth get mon. -o /etc/ceph/ceph.mon.keyring

1. 通过命令获取monitor map：

通过命令获取monitor map后可是实现mon1和mon2、mon3间的通信。命令如下：

ceph mon getmap -o /etc/ceph/monmap

1. 将mon2服务器和mon3服务器添加到monmap：

对于mon2服务器，使用下面命令：

monmaptool --add mon2 192.168.100.12 /etc/ceph/monmap

对于mon3服务器，使用下面命令：

monmaptool --add mon2 192.168.100.13 /etc/ceph/monmap

1. 初始化mon2服务器和mon3服务器的数据目录：

对于mon2和mon3操作相似，做相应变动即可，下面以mon2为例说明下如何初始化mon2的数据目录。对于命令如下：

ceph-mon -i mon2 --mkfs --monmap /etc/ceph/monmap --keyring /etc/ceph/ceph.mon.keyring

1. 将mon2服务器和mon3服务器添加到集群中：

在mon2服务器上：

ceph mon add mon2 192.168.100.12

在mon3服务器上：

ceph mon add mon3 192.168.100.13

1. mon2服务器和mon3服务器的初始化

以mon2服务器为例讲述在扩展服务器初始化的操作，执行下面命令，来完成对应节点初始化：

touch /var/lib/ceph/mon/ceph-mon2/done

以mon2服务器为例讲述扩在展监控服务器上，执行下面命令，来用以表明服务器是通过sysvinit方式启动的：

sudo touch /var/lib/ceph/mon/ceph-mon2/sysvinit

1. 启动mon2服务器和mon3服务器节点

以mon2服务器为例讲述如何启动mon2服务器节点。通过下述命令启动mon2服务器监控：

/etc/init.d/ceph start mon.mon2

1. 修改mon2服务器和mon3服务器，使其ceph服务随系统启动而启动：

同时在mon2服务器和mon3服务器上执行下面命令，完成ceph自启动的设置：

chkconfig ceph on

到此，ceph的搭建已完成，ceph的基本功能已经能够在各个服务器间进行通信。其网络图如图5.1所示：

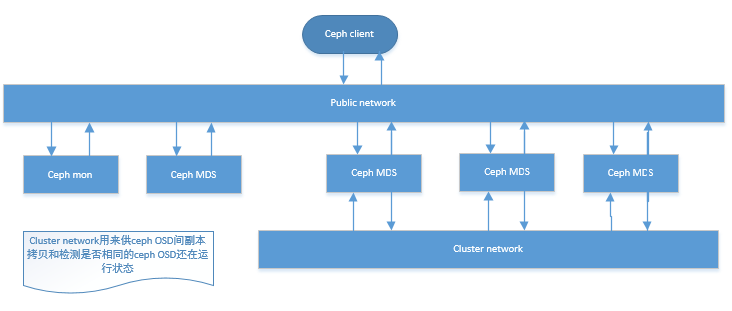


图5.1 ceph网络结构图

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：

# > git clone https://github.com/ceph/calamari.git

# > git clone https://github.com/ceph/calamari-clients.git

# > git clone https://github.com/ceph/Diamond

1. 建立calamari server的rpm包

# > cd calamari

# > yum remove prelink

# > ./build-rpm.sh

1. calamari server的安装

# > cd .. //从calamari目录退出到父目录

# > yum localinstall rpmbuild/RPMS/x86\_64/calamari-server-1.3.1.1-101\_g945d16a.el6.x86\_64.rpm

1. calamari client的安装

在安装calamari client前，需要安装calamari client的依赖包

# > yum install npm ruby rubygems ruby-devel

# > npm install -g grunt grunt-cli bower grunt-contrib-compass

# > gem update --system && gem install compass

下面正式安装calamari client

# > cd calamari-clients

# > make build-real

# > make dist

# > cd ..

# > tar -zxvf calamari-clients\_1.2.2.tar.gz

# > mkdir -p /opt/calamari/webapp/content

# > cd calamari-clients-1.2.2

# >for dir in manage admin login dashboard

do

mkdir -p /opt/calamari/webapp/content/"$dir"

cp -pr "$dir"/dist/\* /opt/calamari/webapp/content/"$dir"/

done

通过以上步骤，calamari server和calamari client的安装基本完成，下面进行calamari的初始化

1. calamari初始化：

# > calamari-ctl initialize

1. 在calamari server端配置防火墙：

### for salt-master

# > iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 4505 -j ACCEPT

# > iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 4506 -j ACCEPT

### for carbon

# > iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 2003 -j ACCEPT

# > iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 2004 -j ACCEPT

1. salt-minion和diamond的安装

diamond rpm包的构建

# > cd Diamond

# > git checkout origin/calamari

# > make rpm

通过执行下面命令，在ceph节点上安装相应的软件包

# > yum localinstall diamond-3.4.67-0.noarch.rpm

# > yum install -y salt-minion

1. 在ceph节点上配置salt-monion并启动

# > touch /etc/salt/minion.d/calamari.conf

###calamari-server-name为calamari 服务器的地址（ip或域名）

# > echo "master: {calamari-server-name}" > /etc/salt/minion.d/calamari.conf

### ：与后面的地址间有个空格

# > echo "master: {calamari-server-name}" >> /etc/salt/minion

# > service salt-minion restart

# > service diamond start

到此，calamari的部署已全部完成。

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，我就不再去详细讲述搭建过程了，仅做简要概述搭建过程。

使用springMVC框架使得业务中视同、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jar包，这样只需要写入我们需要的jar包，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，如要假如spring mvc所需要的jar包。我们还需要创建一个或多个spring的配合文件，这个文件可以实现对bean的管理，也可以做其他用作其他配置。常见好配置文件后，在搭建的项目里要将这个文件包含进去，使得在项目启动时，能加载这个spring的配置文件。

到此，springMVC的最简单的环境就算是搭建好了，可以根据自己的需求，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/上下载和自己操作系统匹配的nodejs版本，并安装。安装nodejs的过程中，npm也会被一起安装。

1. 安装bower：

安装bower需要用到上面安装nodejs过程中安装的npm工具，执行下面命令来安装bower：

npm install –g bower

可用bower工具来管理web应用中的软件包。

1. 安装gulp：

npm install –g gulp

用gulp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

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是能做到协调工作，这就需要有一个管理工具去管理整个项目，控制整个项目的开发过程。我所在的项目组使用的代码版本管理工具是git。下面简单讲述下git版本控制软件在代码开发中的应用。

1. git安装：

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

1. 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

1. 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，这个区去想你最后提交的代码版本。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。

1. 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，需要通过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

1. 分支的使用：

在代码的开发过程中，分支最常被用在做特性开发。使用分支，可以减少对代码master分支上的维护，是的每个小组只需要维护他们所创建的分支即可，有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。当小组将对应的特性开发完成，在一次将对应的分支合并到主线上，通过这个过程，分支实现了自己的特性开发全过程。通过git checkout –b “分支名称”命令实现分支的创建，在创建分之后，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之的，只有使用git push origin < branch >才能将本地分之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之。

1. 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，去更新下本地代码，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，就不会出现刚才所说的冲突。冲突出现在有两个和两个以上的人在修改同一个文件。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了冲突文件。修改后，通过命令git add < filename >命令，标记解决冲突成功。

1. 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，是的本地HEAD区的项目无法正常运行起来，这是你想回到上个HEAD版本，你可以通过命令git chechout -- < filename >来挽回刚才的误操作，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实现想本地代码指向最新过去的代码。

5.4 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。使得个项目的结构清晰，升级维护方便，代码利用率高。通过分布式版本控制工具git，为各个开发小组提供了极大的便利。

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 环境的搭建

1. 项目的克隆：

使用git的工具，克隆远程服务器段项目。

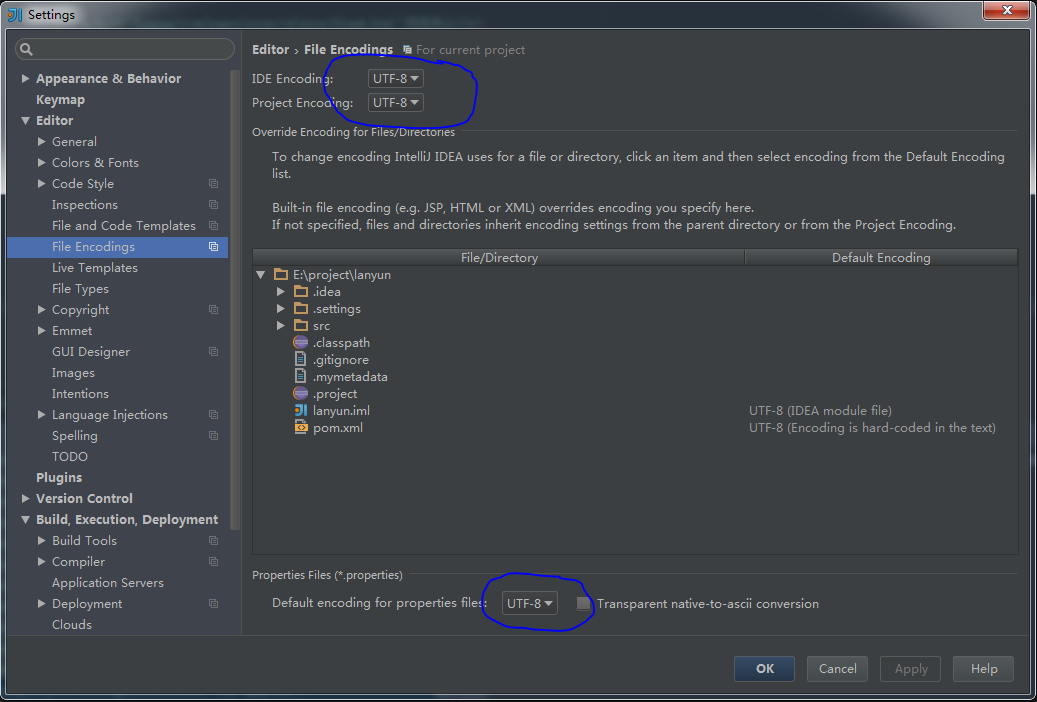
1. 创建工程：
2. 在IEDA官网：<http://www.jetbrains.com/idea/download/> #section=windows下载IDEA开发工具，并安装。
3. 打开IDEA，点击File->Setting选项，配置IDEA默认编码方式如下图6.1所示：

图6.1 设置IDEA默认编码方式

1. 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：

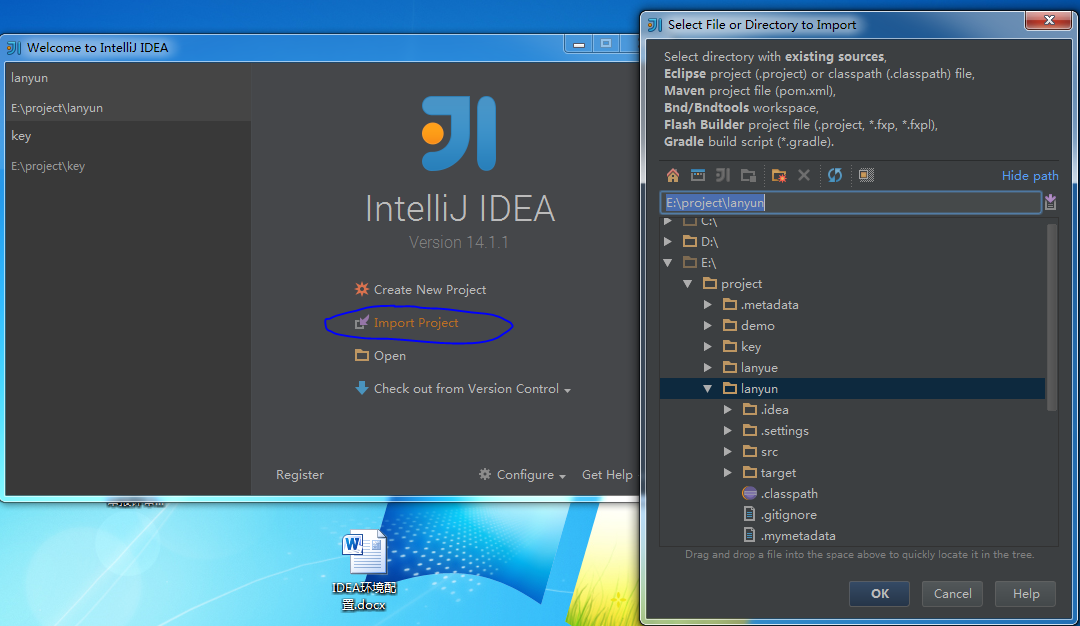


图6.2 添加项目

1. 选中所选项目，点击OK，选择maven模式，如图6.3所示：

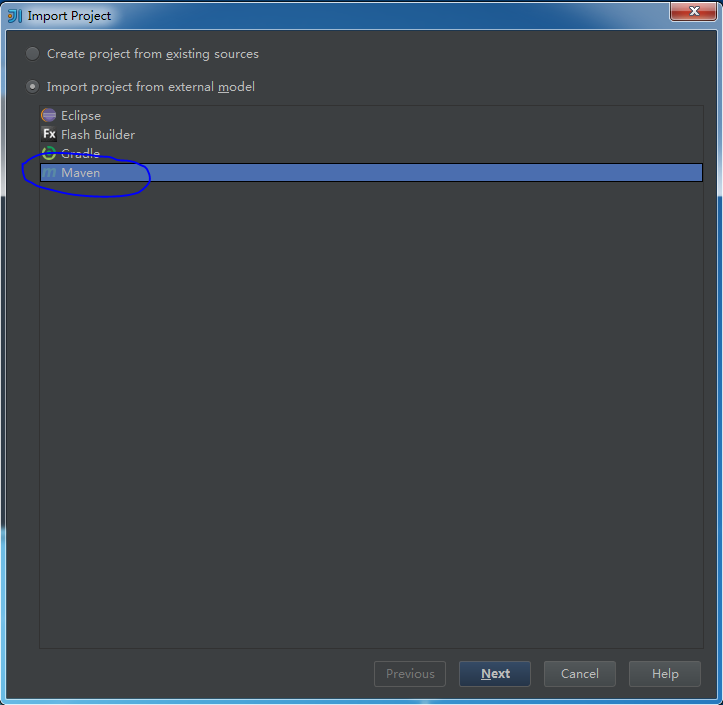


图6.3 maven项目选取

1. 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：

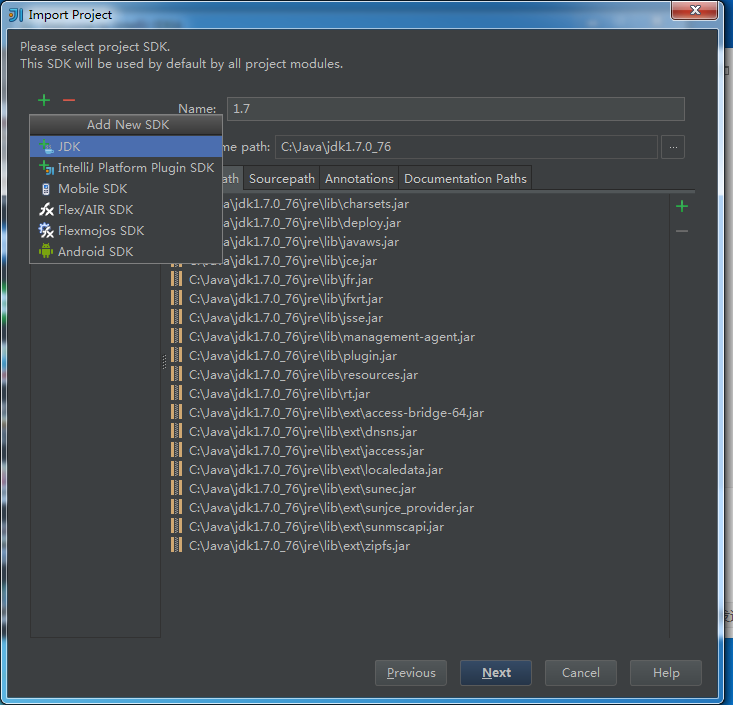


图 6.4 Please select project SDK界面

1. 点击Next直至完成。到此，项目的常见工作就已经全部完成
2. 运行环境的配置：

在（2）中已将将要开发的项目导入了IDEA开发工具中，现在通过配置运行环境是的项目能够正常运行起来。

1. 在IDEA界面，点击File->Setting选项，通过下图的选项，选择Bulid 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：

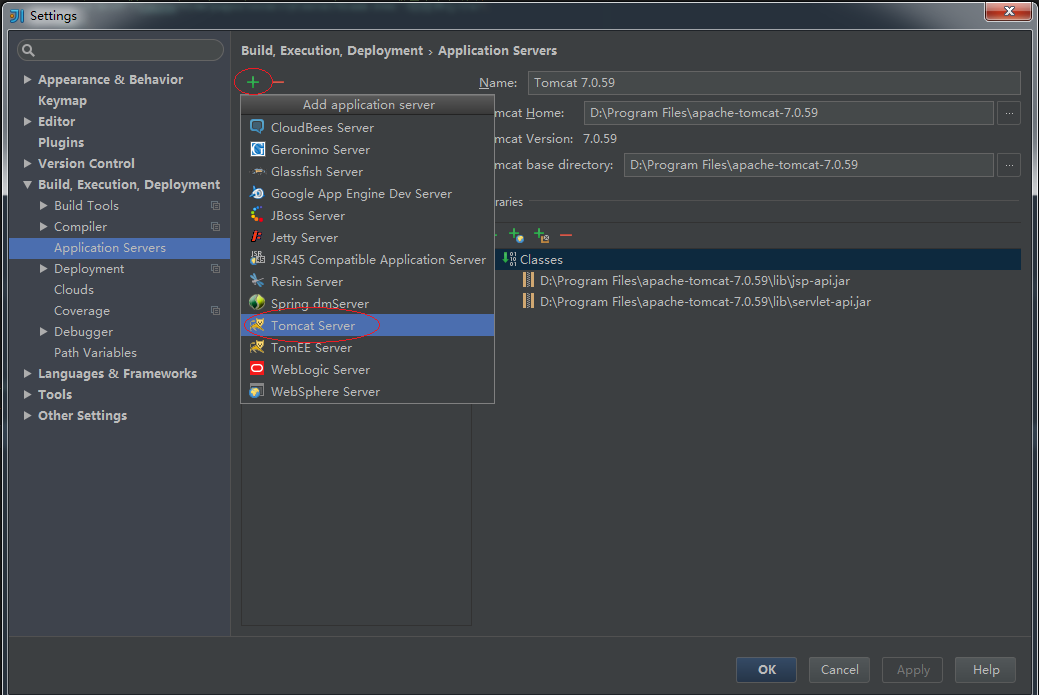


图6.5 Tomcat Server选项卡

1. 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：

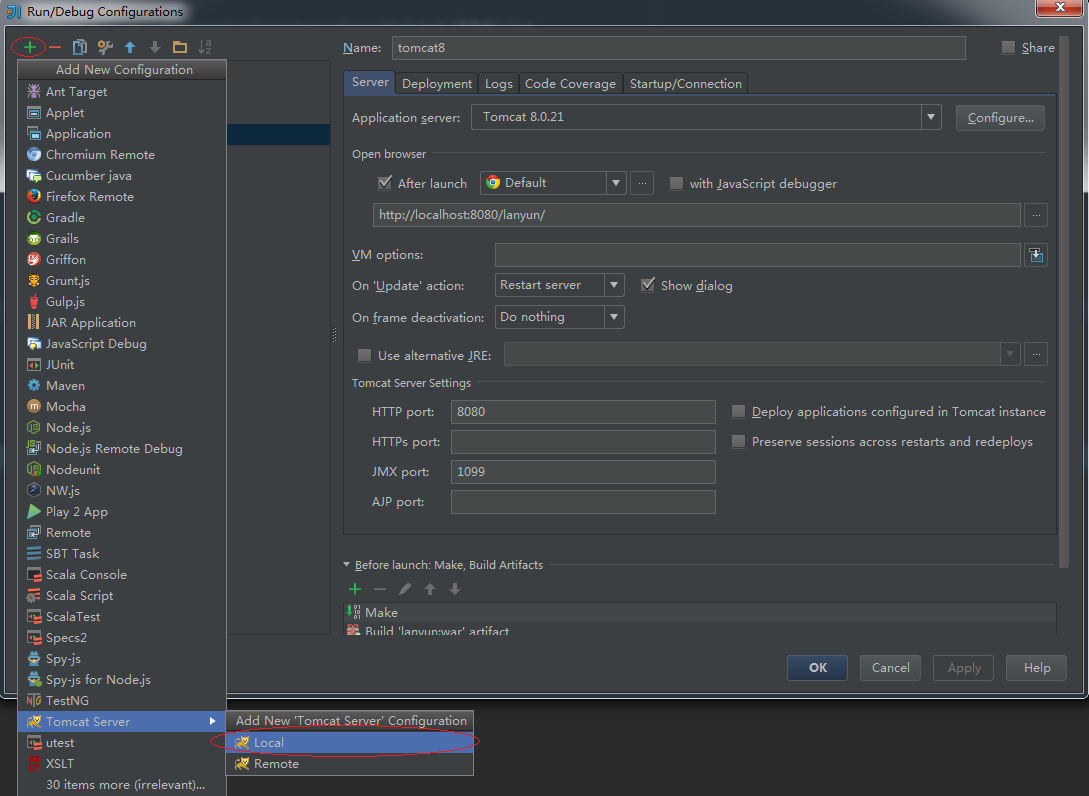


图6.6 Tomcat Server设置

1. 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指用于显示在运行选项时对应的值。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。

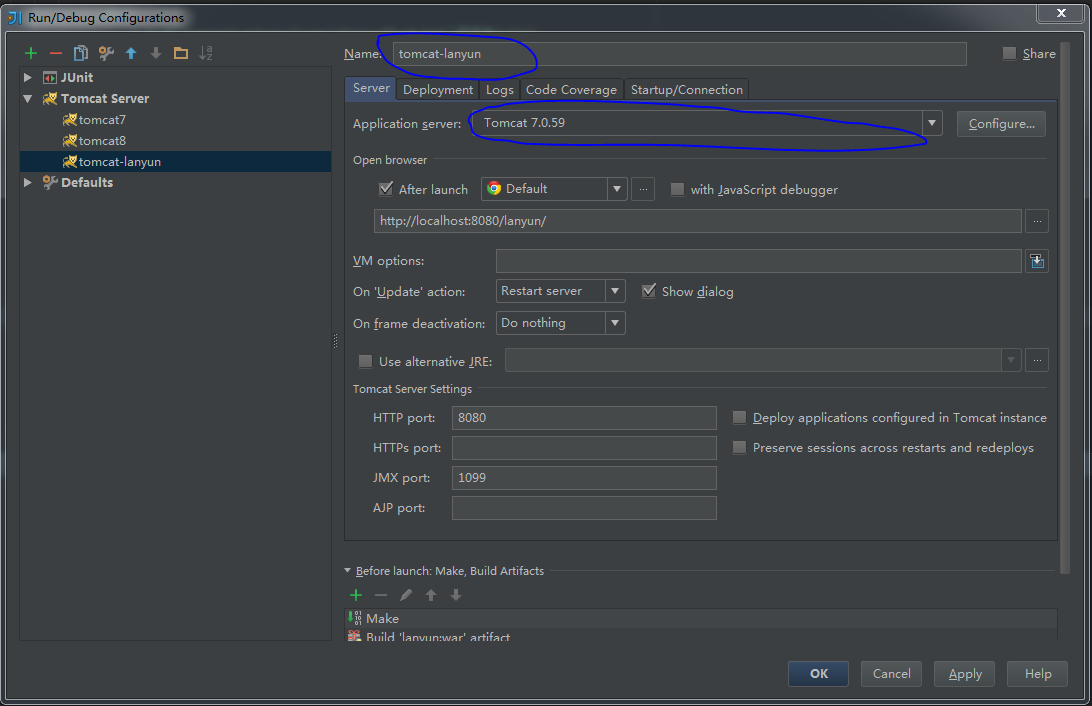


图6.7 Server选项的配置

1. 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。

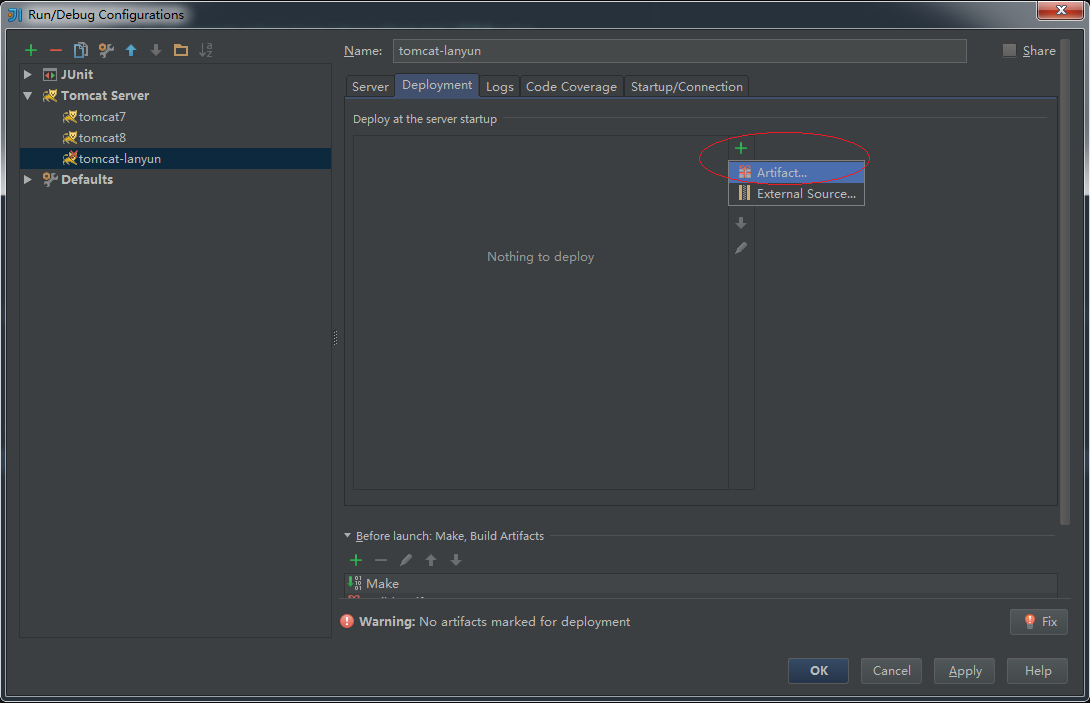
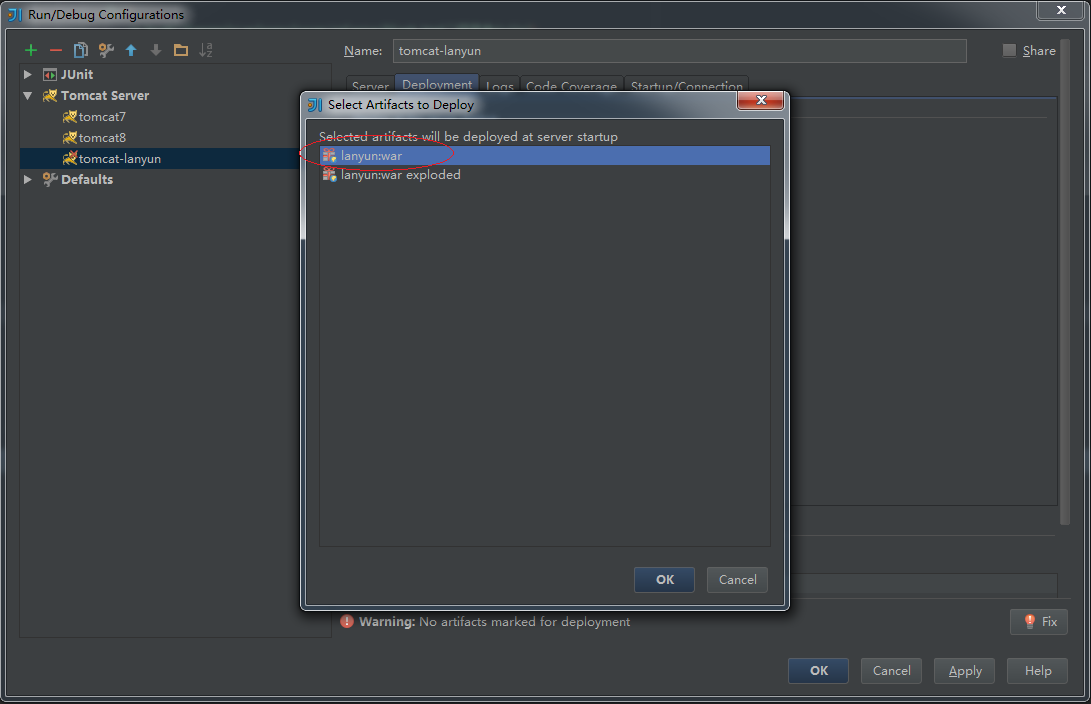


图6.8 Deployment选项

1. 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。



6.9 Artifact选项

1. 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：

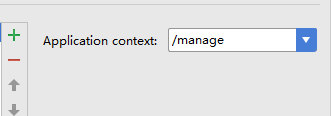


图6.10 路径设置

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列表信息，可以通过每个集群的详细信息窗口可以查看每个对应集群的信息概览，在众多信息概览里要突出主要信息，及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。具体活动图如下：

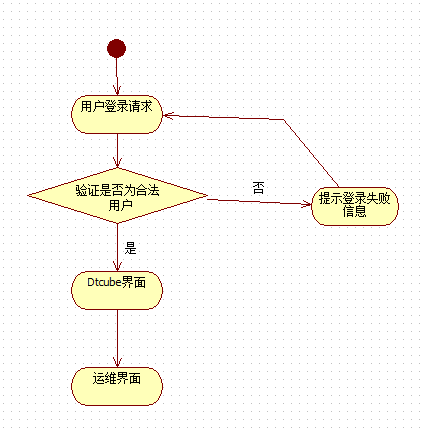
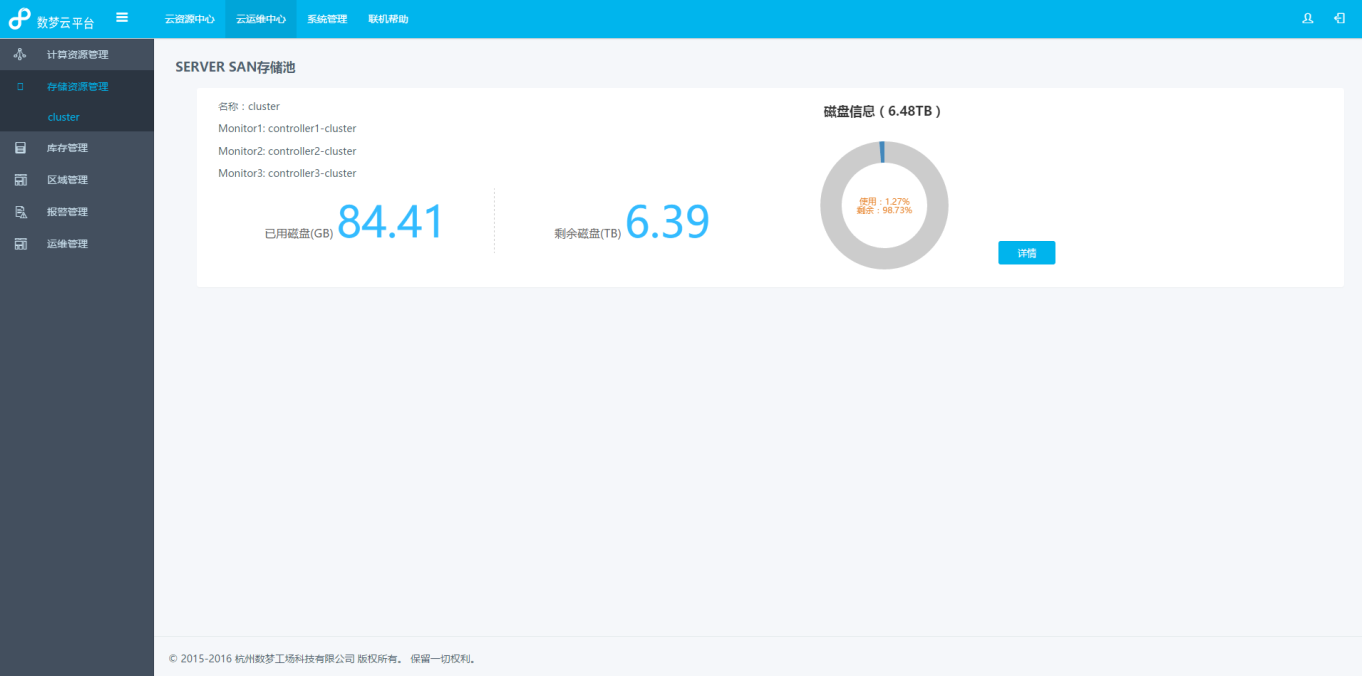


图6.1 用户活动图

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6.2所示，显现了集群列表信息。



6.2 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，我们可以查看此集群的详细信息，可以进入集群详细界面，如下图6.3所示：

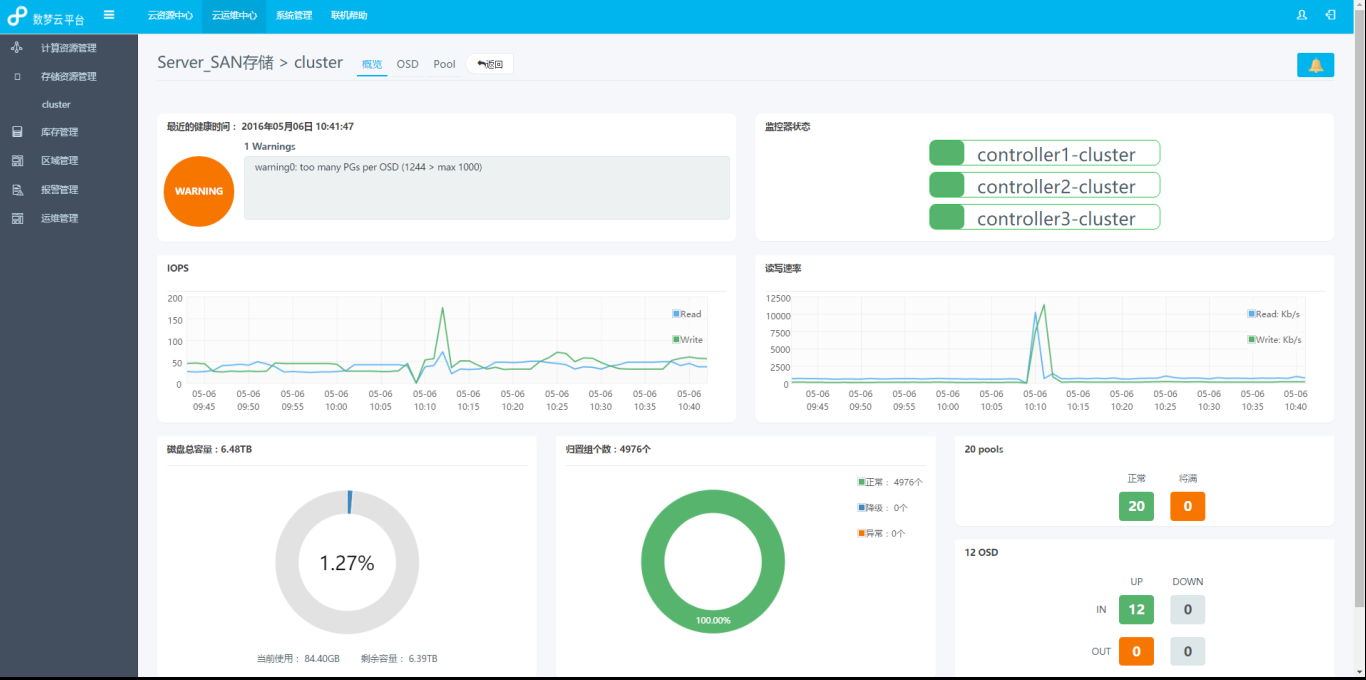


图6.3 集群信息概览

通过图6.3，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6.4是OSD界面用形象的展示机架上服务器和OSD布局情况。

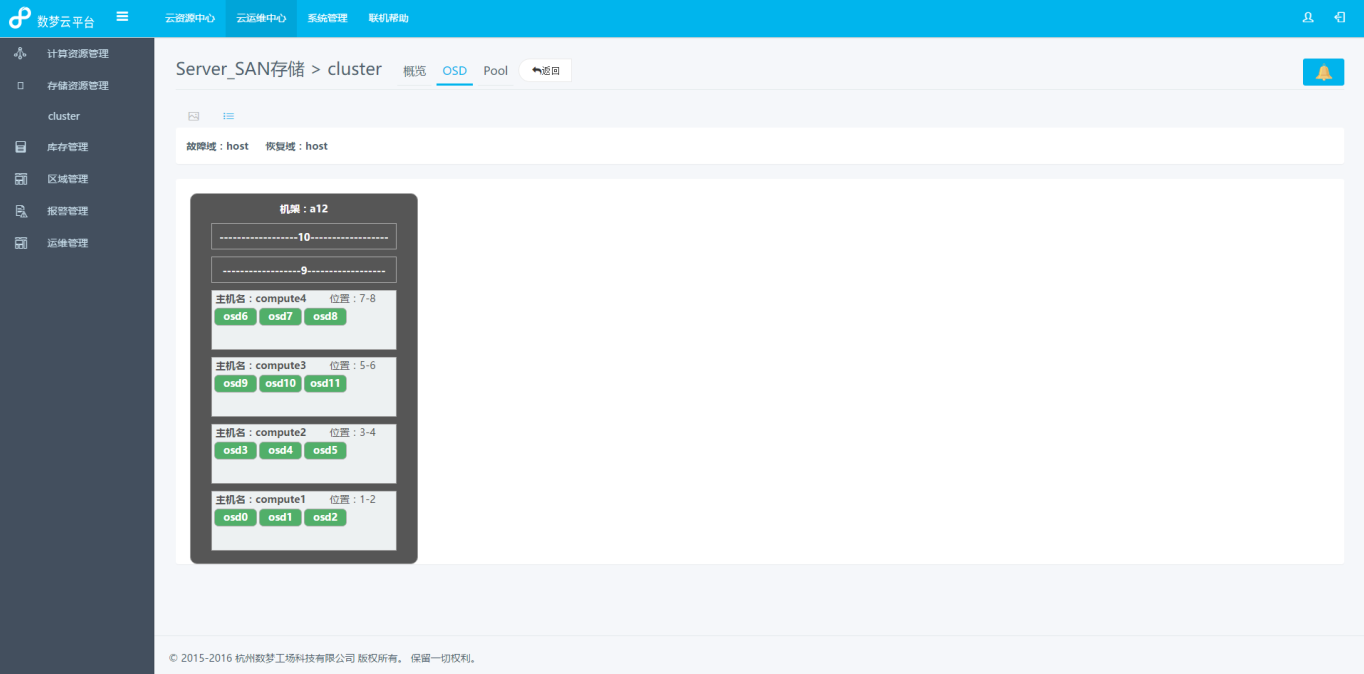


图6.4 OSD在机架上的布局

通过图6.4我们可以看到这个集群所有OSD所在的机架和对应的服务器。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6.5是通过点击机架上OSD8所得到的关于每个OSD所对应的信息表。

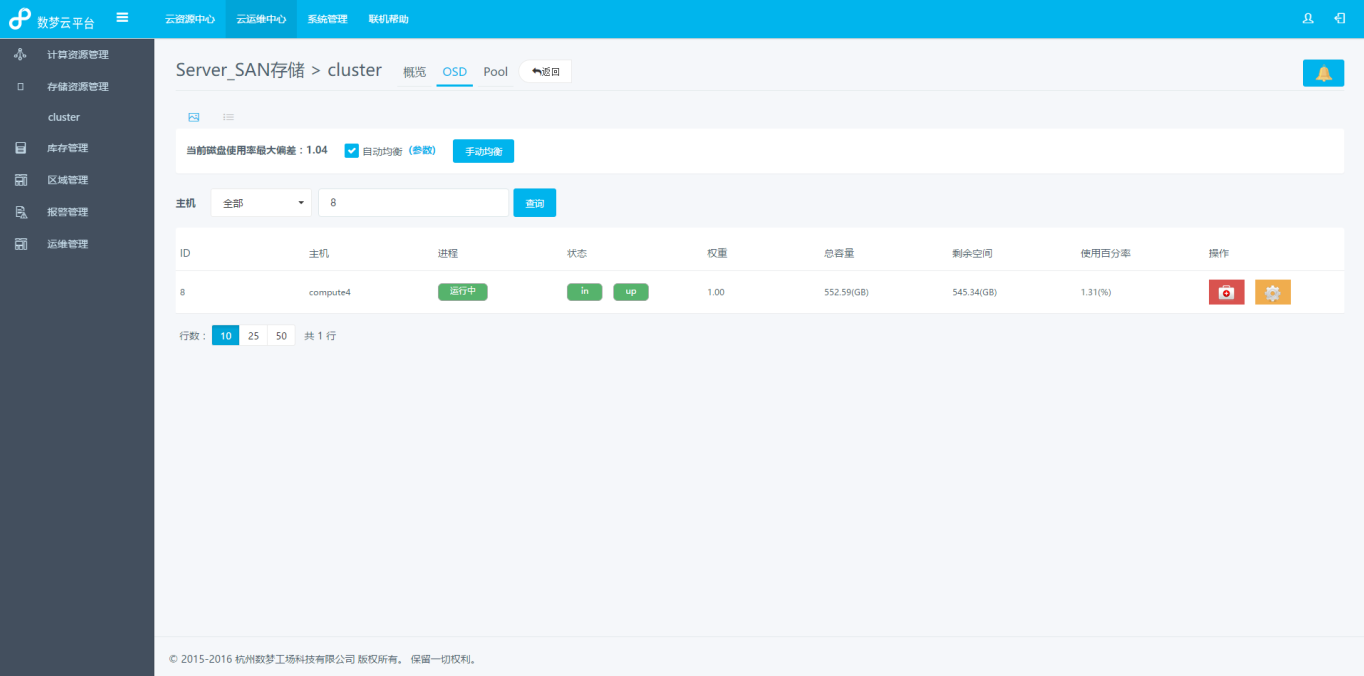


图6.5 OSD详细列表

在这里我们可以看到每个OSD所对应的详细信息，同事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6.6所示：

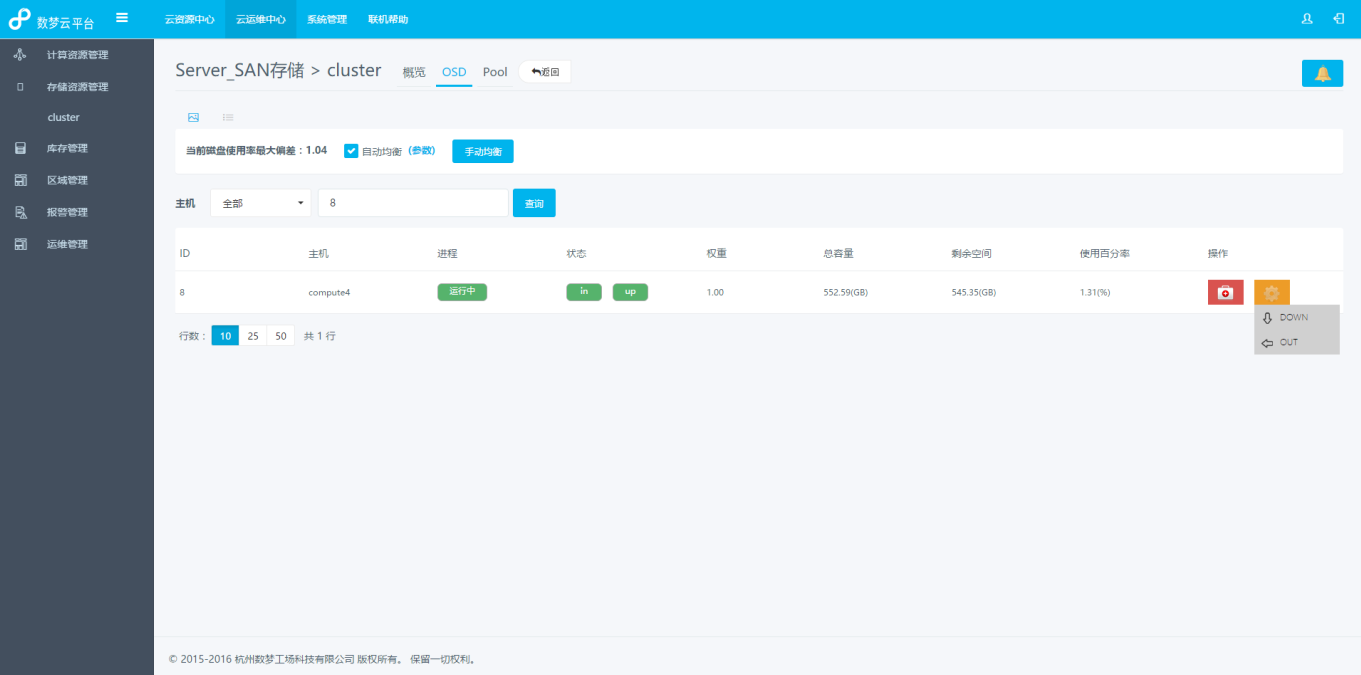


图6.6 OSD设置

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，其中包括手动均衡和自动均衡。手动均衡界面如图6.7所示：



图6.7 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是，难免会出现数据在磁盘分布不均的情况，或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，可以通过自动均衡选项，让系统在规定的时间内做均衡操作，这样可以最大限度的避免磁盘均衡时给用户带来的不便。自动均衡界面如6.8所示。



图6.8 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，每个Pool所拥有的副本，每个Pool所具有的pg个数，每个Pool的磁盘大小和已经使用过的磁盘大小。详细信息如图6.9所示。

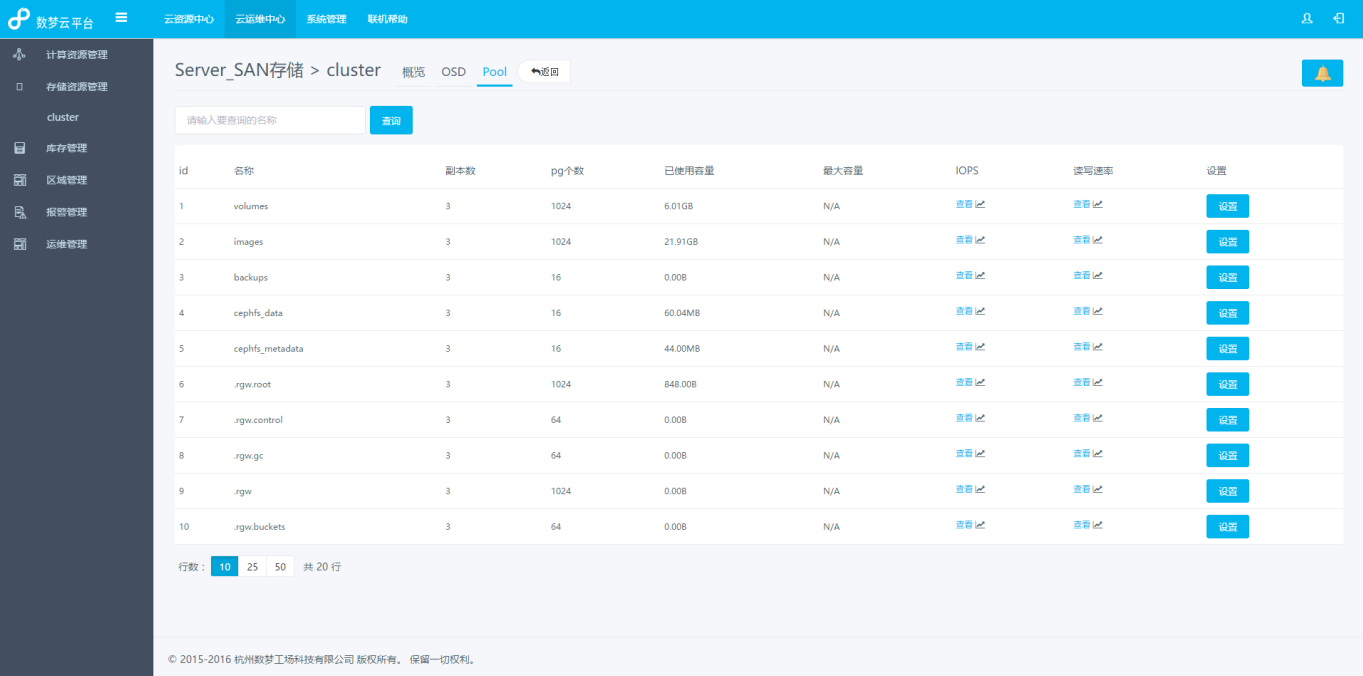


图6.9 Pool的详细列表

在详细列表中有IOPS，一般用于表示Pool磁盘中，当数据随机读或随机写时所对应的值，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在近一段是内的值得变化，可以通过图6.10相信展示出IOPS对应的数值信息。

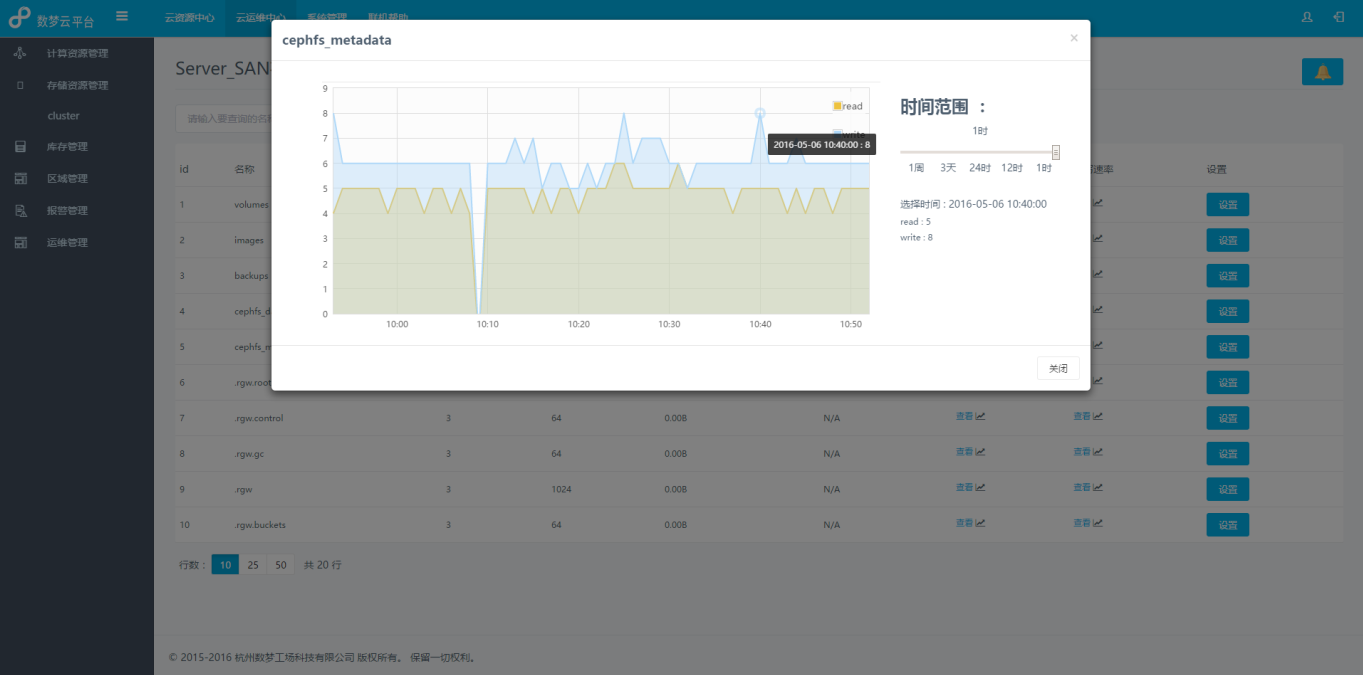
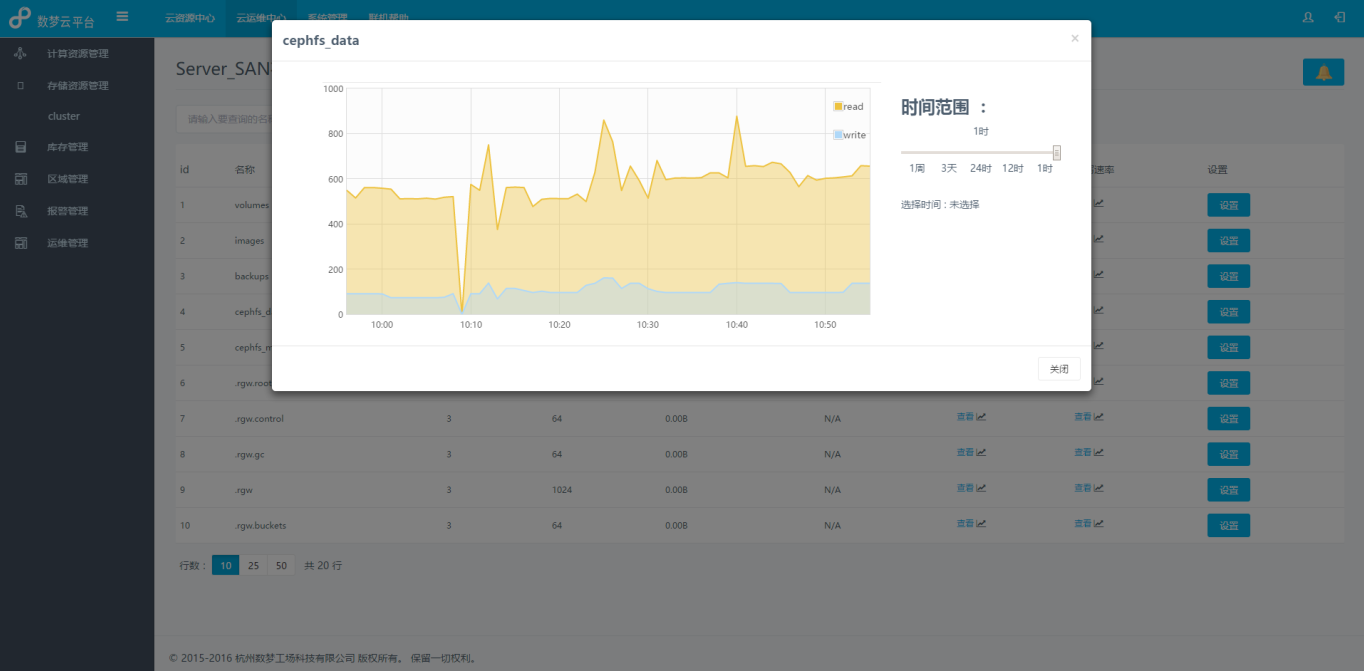


图6.10 cephfs\_metadata的IOPS的信息

通过6.10，我们可以看到IOPS对应的信息，其中单位用次/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6.11所示。



6.11 cephfs\_datade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6.12可以将这部分的信息展现出来。



6.12 Pool volumees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，前端界面的现在所实现的功能几乎就这么多了，随着后期的需求，会在相应的界面添加对应的功能模块。

1. 后台功能的实现

在原本的框架中，前端后台Java交互的是利用的angularjs中的$resource模块，利用$resource模块，利用制定的api字符串，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机制实现更灵活的处理异步函数的调用。

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，通过@Controller标记，系统会找到对应的java代码中的用作控制器模块的java代码，在通过@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，同事处理从更底层返回的数据，对数据进行拆分组合，从中提取用户需要的信息，去掉用户不感兴趣的信息，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，我们成为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，通过调用CloseableHttpClient类的execute()方法，通过配置文件实现和底层的calamari的交互，在有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，这些时关于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、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。接下来讲述下实习过程中遇到的较为典型的，困惑我好久，需要添加新模块才解决问题。

1. 数据请求延时：
2. 问题描述：

在云存储运维系统模块中，有时从后台取数据需要很长的时间。数据获取时间的长短和后台维护的磁盘容量成正比，当磁盘容量很大是，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。而在另外的界面就回去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，即使正常的切换界面速度，在页面数据信息显示时，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卡顿，有时还会出现卡死状态。

1. 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，会发现，当采用项目框架的方法，用$resource模块中的get( function ( res ) )往后发发送请求的方法时，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。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切面切换，就会有很多无用过得js文件继续停留在内存，当对应的数据返回后，有相应接受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，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，在去和其他无用的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接界面时，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价容易了。当有用的js文件获得处理器的使用权时，去处理数据的，当遇到异步操作是，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，这个会使得当前js线程失去处理器使用权，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了。刚开始对这个现象确实感到困惑，这个现象还不容易复现，所以使我困惑了好久。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出现。我就是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，同是还会和前天界面进行交互。如果恰巧无用的js线程在做数据处理时，调用了前台的标签所数据处理，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。

1. 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。我就希望能通过新的数据传输方法去实现数据的调用。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$destory事件，通过在$destory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情。利用在XMLHttpRequest模块中，具有立刻终止数据请求的函数，结合Promise实现机制，可以在界面发生跳转前，使得发送数据请求的部分，可以得到相应的回应。这样实现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。在跳转到新的界面事，上个界面的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。

1. 浏览器间兼容性：
   1. 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。

* 1. 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。

* 1. 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。后来，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。

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。当时出现过，我花费了一天的时间去将我开发的代码何如远程我们开发分支。这种手动何如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，以为我这种手动何如的方式耗时太长，多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。

由于每次何如代码，都是手动检查异同。当时总是感觉，代码合入远程分支，时间非常复杂和耗时的一件事。后来通过同事聊天，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。通过咨询再知道正确的合并代码的方法。通过这个事件，是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。同事也展现了和同事交流的重要性。

在整个实习的过程中，有收获，有感慨也有成就感。在实习的过程中，我感知到了基础知识的重要。在实习的过程中，学会了很多，是我更加坚信，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

附录

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我了极大的启发和帮助。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给出了很大的帮助。感谢母校对我的培养和展现自我的机会。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，我们在不断的探讨和研究的过程中，给与了我很多帮助，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疑难的问题，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写作的思路。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了基石。

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。