# 基于Spring Cloud的异构数据库管理平台

# 1 背景知识

1 Spring Cloud简介

要了解Spring Cloud，首先了解Spring Boot。

Spring Boot简化了基于Spring的应用开发，通过少量的代码就能创建一个独立的、产品级别的Spring应用。 Spring Boot为Spring平台及第三方库提供开箱即用的设置，这样你就可以有条不紊地开始。多数Spring Boot应用只需要很少的Spring配置。

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。用我的话来理解，就是Spring Boot其实不是什么新的框架，它默认配置了很多框架的使用方式，就像maven整合了所有的jar包，Spring Boot整合了所有的框架。

Spring Boot的核心思想就是约定大于配置，一切自动完成。采用Spring Boot可以大大的简化你的开发模式，所有你想集成的常用框架，它都有对应的组件支持。

进入正题，下面介绍Spring Cloud。

Spring Cloud是一系列框架的有序集合。它利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部署。Spring并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过Spring Boot风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

微服务是可以独立部署、水平扩展、独立访问（或者有独立的数据库）的服务单元，Spring Cloud就是这些微服务的大管家，采用了微服务这种架构之后，项目的数量会非常多，Spring Cloud做为大管家就需要提供各种方案来维护整个生态。

Spring Cloud就是一套分布式服务治理的框架，既然它是一套服务治理的框架，那么它本身不会提供具体功能性的操作，更专注于服务之间的通讯、熔断、监控等。因此就需要很多的组件来支持一套功能。

Spring Cloud与Spring Boot之间具体关系如何呢？Spring Boot是Spring的一套快速配置脚手架，可以基于Spring Boot 快速开发单个微服务，Spring Cloud是一个基于Spring Boot实现的云应用开发工具；Spring Boot专注于快速、方便集成的单个微服务个体，Spring Cloud关注全局的服务治理框架；Spring Boot使用了默认大于配置的理念，很多集成方案已经帮你选择好了

2 Docker 简介

Docker 是世界领先的软件容器平台。开发人员利用 Docker 可以消除协作编码时“在我的机器上可正常工作”的问题。运维人员利用 Docker 可以在隔离容器中并行运行和管理应用，获得更好的计算密度。企业利用 Docker 可以构建敏捷的软件交付管道，以更快的速度、更高的安全性和可靠的信誉为 Linux 和 Windows Server 应用发布新功能。

Docker 属于 Linux 容器的一种封装，提供简单易用的容器使用接口。它是目前最流行的 Linux 容器解决方案。Docker 将应用程序与该程序的依赖，打包在一个文件里面。运行这个文件，就会生成一个虚拟容器。程序在这个虚拟容器里运行，就好像在真实的物理机上运行一样。有了 Docker，就不用担心环境问题。

总体来说，Docker 的接口相当简单，用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。

Docker 最初是 dotCloud 公司创始人 Solomon Hykes 在法国期间发起的一个公司内部项目，它是基于 dotCloud 公司多年云服务技术的一次革新，并于 2013 年 3 月以 Apache 2.0 授权协议开源，主要项目代码在 GitHub 上进行维护。Docker 项目后来还加入了 Linux 基金会，并成立推动 开放容器联盟（OCI）。

Docker 自开源后受到广泛的关注和讨论，至今其 GitHub 项目已经超过 4 万 6 千个星标和一万多个 fork。甚至由于 Docker 项目的火爆，在 2013 年底，dotCloud 公司决定改名为 Docker。Docker 最初是在 Ubuntu 12.04 上开发实现的；Red Hat 则从 RHEL 6.5 开始对 Docker 进行支持；Google 也在其 PaaS 产品中广泛应用 Docker。

下面详述一下Docker的优势：

容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行 10 个不同的应用就要起 10 个虚拟机，而Docker 只需要启动 10 个隔离的应用即可。具体有如下优势：

1、更快速的交付和部署

对开发和运维（devop）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。 Docker 可以快速创建容器，快速迭代应用程序，并让整个过程全程可见，使团队中的其他成员更容易理解应用程序是如何创建和工作的。 Docker 容器很轻很快！容器的启动时间是秒级的，大量地节约开发、测试、部署的时间。

2、更高效的虚拟化

Docker 容器的运行不需要额外的 hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效率。

3、更轻松的迁移和扩展

Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。 这种兼容性可以让用户把一个应用程序从一个平台直接迁移到另外一个。

4、更简单的管理

使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

# 2 系统架构

本系统的框架如下图所示



数据库容器管理服务负责管理docker数据库容器集群，包括数据库容器的启动、关闭、删除。该服务向服务注册中心注册服务后，消费者通过服务注册中心获得服务列表。从而根据获得的服务名调用服务。

# 3 实验

## 3.1 实验环境

|  |  |
| --- | --- |
| 工具 | 版本 |
| Java | Jdk 1.8 |
| Spring Cloud | 1.3.5 |
| Docker | 18.03 |
| Maven | 3.5 |

## 3.2 实验步骤

1.docker 创建数据库

我们首先通过docker创建多个异构数据库，为了部署方便，此处采用DockerFile的形式完成docker容器的部署创建。文件如下

2 搭建服务注册中心

由于Spring Cloud是在Spring Boot的基础上实现的，所以我们首先创建Spring Boot工程，并采用maven方式进行jar包管理。pom.xml中的依赖如下：

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.3.5.RELEASE</version>  
 <relativePath/>  
</parent>  
  
<properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <java.version>1.8</java.version>  
</properties>  
  
<dependencies>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-eureka-server</artifactId>  
 </dependency>  
  
</dependencies>  
  
<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Brixton.RELEASE</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

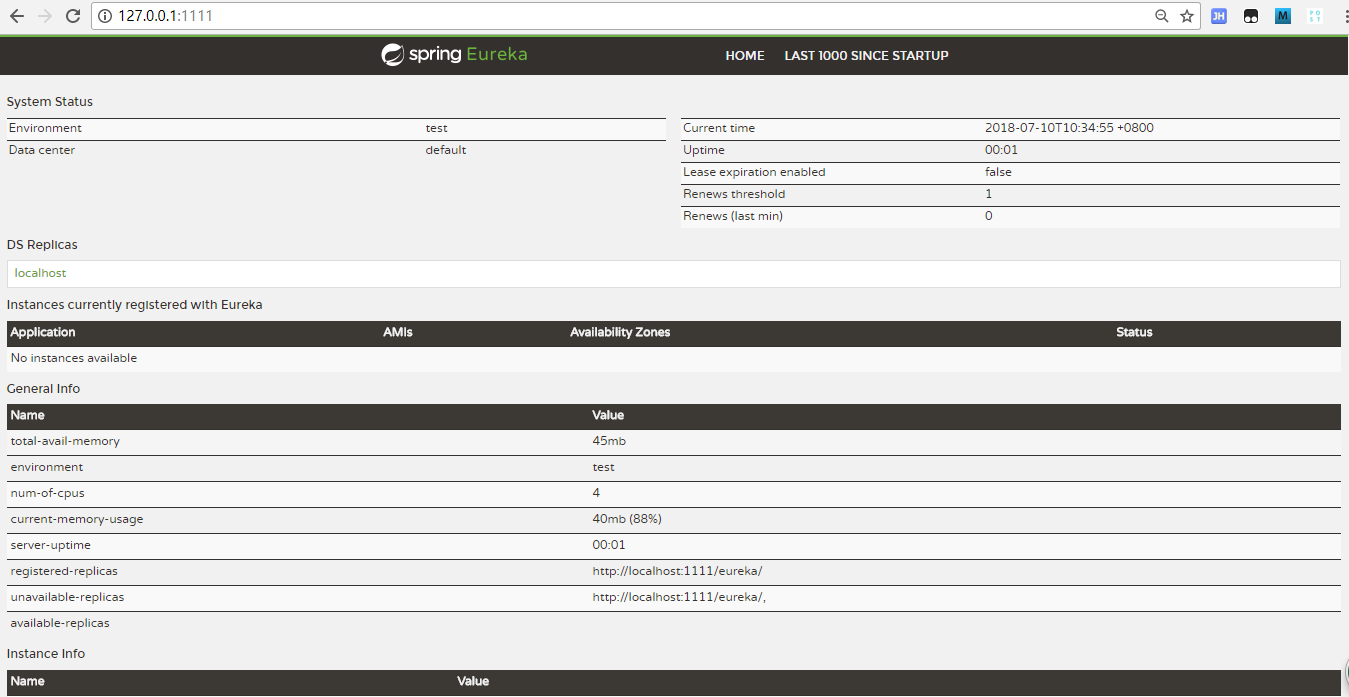
之后在Spring Boot主程序中添加@EnableEurekaServer注解启动服务注册中心。

import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.builder.SpringApplicationBuilder;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@EnableEurekaServer  
@SpringBootApplication  
public class Application {  
 public static void main(String[] args) {  
  
 new SpringApplicationBuilder(Application.class).web(true).run(args);  
 }  
  
}

由于服务注册中心往往也会将自己注册为服务，此处我们要禁用该行为，在application.properties中增加如下配置。

server.port=1111  
eureka.client.register-with-eureka=false  
eureka.client.fetch-registry=false  
eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/

其中服务注册中心端口为1111，不检索服务，不向注册中心注册自己。



访问127.0.0.1可看到如下面板，其中Instances currently registerd with Eureka一栏为空，表明未注册任何服务。

3 注册服务提供者

相比于服务注册中心，注册服务提供者只多了这些依赖。

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-eureka</artifactId>  
</dependency>

在主类上加上注解@EnableDiscoveryClient,激活Eureka中的DiscoveryClient实现

import org.mybatis.spring.annotation.MapperScan;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
  
@EnableDiscoveryClient  
@SpringBootApplication  
@MapperScan("com.didispace.dao")  
public class HelloApplication {  
  
 public static void main(String[] args) {  
  
 SpringApplication.*run*(HelloApplication.class, args);  
 }  
  
}

最后在application.properties配置文件中进行如下配置：

spring.application.name=hello-service  
  
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/

spring.application.name属性表示服务的名称，eureka.client.serviceUrl.defaultZone表示服务注册中心的地址。

在Controller中实现具体的服务内容

4 服务消费

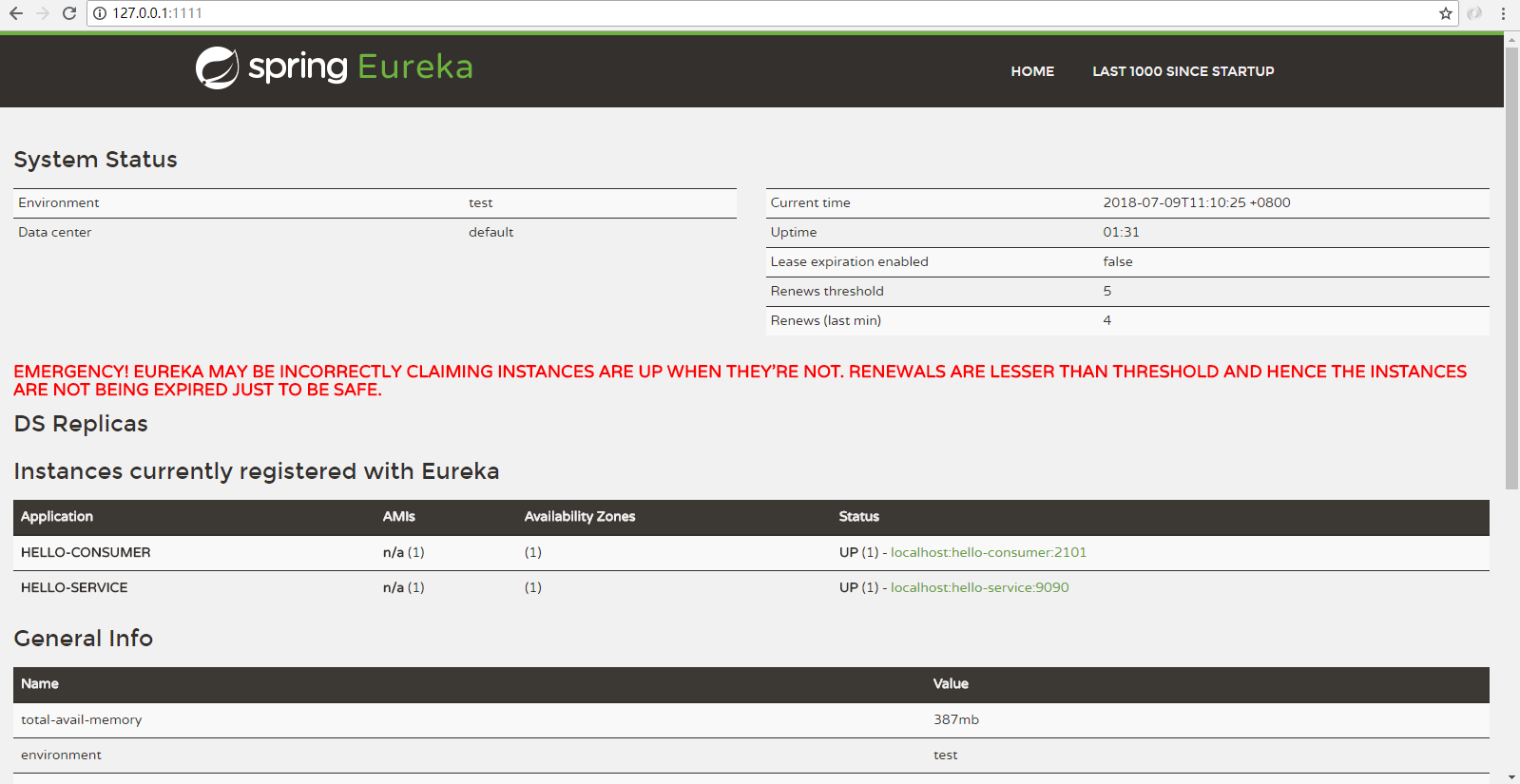
创建应用主类ConsumerApp通过@EnableDiscoveryClient注解让该应用注册为Eureka客户端应用，以获得服务发现的能力。同时在主类中创建RestTemplate的Spring Bean实例，并通过@LoadBalanced 注解开启客户端负载均衡。

import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
import org.springframework.cloud.client.loadbalancer.LoadBalanced;  
import org.springframework.context.annotation.Bean;  
import org.springframework.web.client.RestTemplate;  
  
@EnableDiscoveryClient  
@SpringBootApplication  
public class ConsumerApp {  
 @Bean  
 @LoadBalanced  
 public RestTemplate restTemplate(){  
 return new RestTemplate();  
 }  
 public static void main(String args[]){  
 SpringApplication.*run*(ConsumerApp.class,args);  
 }  
  
  
}

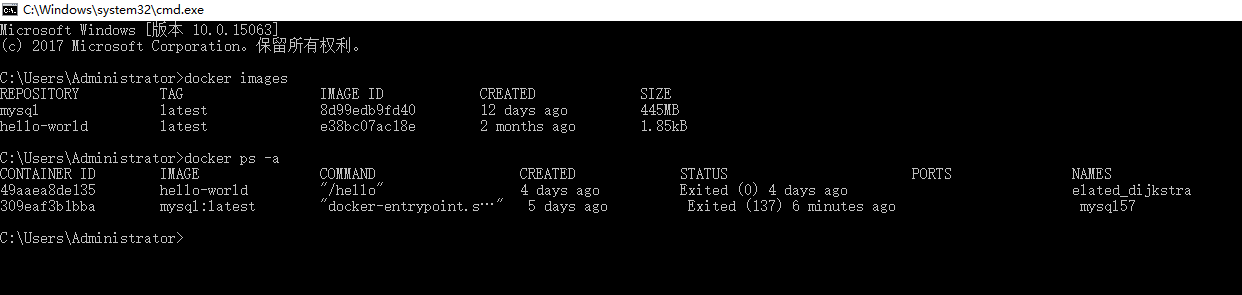
在application.properties中配置同之前的服务提供者类似，但需要改变消费者端口，不能与之前启动的应用端口冲突。配置如下：

spring.application.name=hello-consumer  
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/  
  
server.port=2101  
spring.mvc.view.prefix=/resources/  
spring.mvc.view.suffix=.html

## 3.3 实验结果

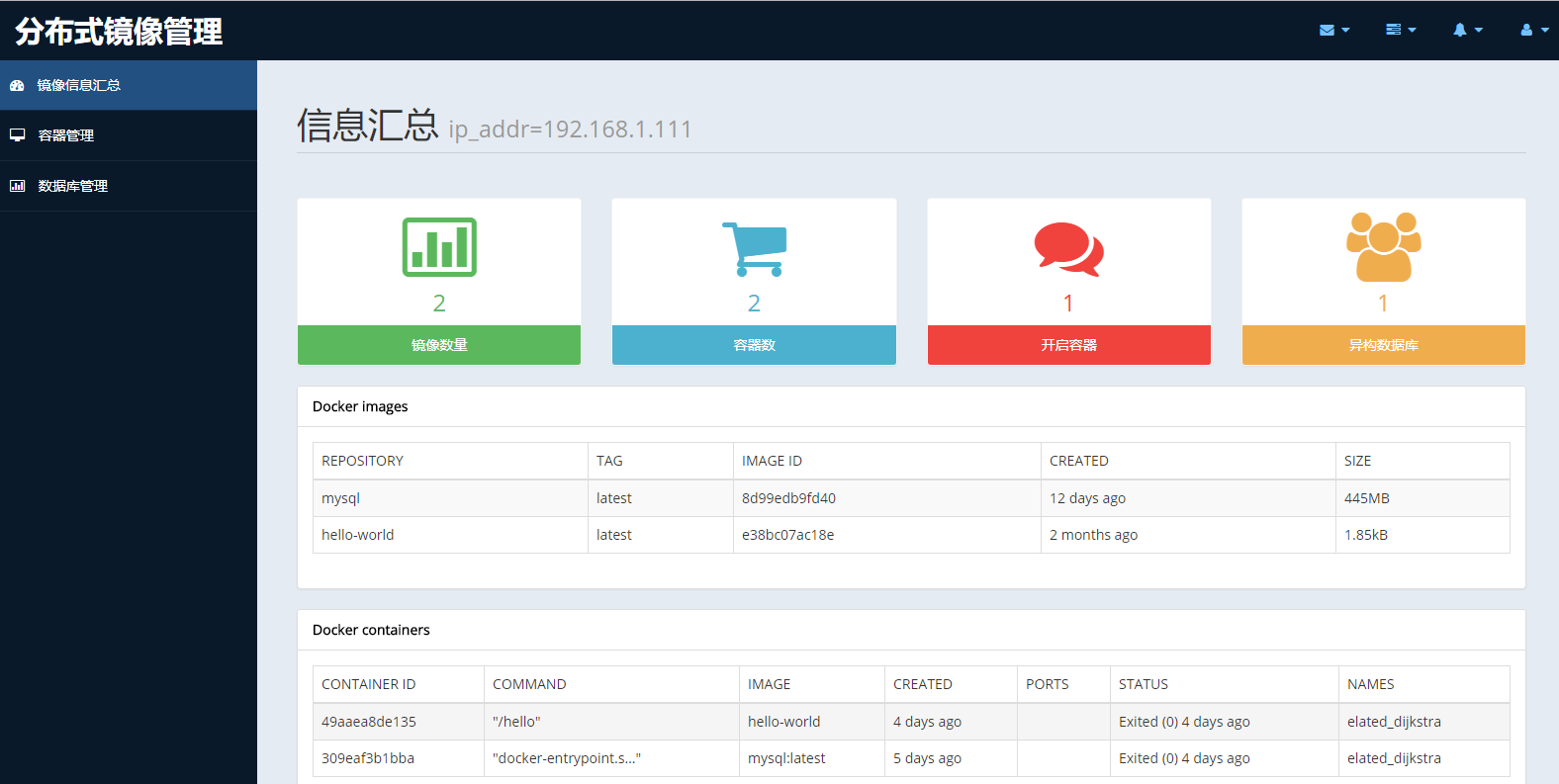


如上图，服务启动后，访问<http://127.0.0.1:1110/>可以看到Eureka的信息面板，我们可以从信息面板中看到最基本的信息，包括当前时间、更新时间等。另外，Instancescurrently registered with Eureka一栏显示注册到服务注册中心内的服务。可以看到，我们的系统注册了两个服务，HELLO-CONSUMER服务和HELLO-SERVICE。顾名思义，HELLO-CONSUMER服务是提供给客户端用户的服务，HELLO-SERVICE服务是系统用于后台管理容器的服务入口。同时，我们还可以在最下面看到系统的大体信息，目前可用的内存空间和当前环境。



我们在客户端打开cmd命令行，可以执行docker相关的命令，如上图，我们首先运行”docker image”列出本地所有的docker镜像。“docker image”输出结果第一行的REPOSITORY是镜像的来源库，TAG是每个镜像都有自己的标签，IMAGE ID是每个镜像独有的ID。同时还有创建时间CREATED和大小信息SIZE。从输出结果中可以看出当前本地有两个镜像，mysql和hello-world镜像，两个镜像的具体信息分别列出在对应的标签列下。

同时，我们可以运行命令“docker ps -a” 显示所有的容器，包括未运行的。该命令能够显示更加详细的信息，包括CONTAINER ID、IMAGE、COMMAND、CREATED、STATUS等，分别对应的是容器的独立id、镜像文件、容器对应的命令、创建时间、状态等信息。除此之外，我们通过运行更多docker命令查看、管理容器和镜像。



如上图是我们分布式镜像管理整个界面，左侧是功能列表，包含镜像信息汇总、容器管理和数据库管理三个大的功能类。当然，这是我们构建的系统雏形，在后续的运行和管理中可以添加更多的功能分类。下面就每个功能具体分析。首先是镜像信息汇总页，默认情况下，分布式镜像管理系统首页就是信息汇总，所以每次打开系统，用户可以快速获取到系统目前管理的内容和运行情况，这是一种极其理智和聪明的做法。在系统显示的右侧面板中，我们可以清晰看到面板被整齐分为三个部分，整齐划一图像链接显示部分，docker镜像信息管理栏，最后是docker容器信息管理栏。第一部分中，显示了4个具体的信息，包含了镜像数量、容器数、当前开启的容器、异构数据库数量。我们可以明了看到目前系统由两个镜像、两个容器、一个正在运行的容器、还有一个异构数据库。第二部分和我们在命令行中运行”docker image”获取到的镜像信息是一样的，我们这里通过表格方，把它简单明了地表示出来。最后一栏是docker容器的具体信息，当然，它也与在cmd中运行”docker ps -a”获取的信息一样，我们这里也收集管理起来了。通过信息汇总页的信息展示，我们可以清晰、方便查看到我们系统当前镜像的全部信息。



上图是我们系统功能的容器管理部分，这个功能可以为系统管理员提供傻瓜式的容器管理方式。在该功能模块中，管理员只需要按照简单的功能指示就可以完成容器的开启、关闭、删除等等功能。由于我们的功能在日以继夜地完善，所以很快将有更多的功能呈现给用户。可以更加方便用户的使用，进一步提升用户的体验。在右侧的显示面板中，我们可以看到目前数据库管理的容器中包含了两个数据库，分别是Mysql数据库和Oracle数据库的管理，具体实现的功能还有开启和关闭的功能。