中国石油大学（华东）
CHINA UNIVERSITY OF PETROLEUM

本 科 毕 业 设 计（论文）

题　目：基于 Docker 的订水管理系统开发运维

学生姓名：王　辉

学　　号：12072226

专业班级：软件工程 12-2 班

指导教师：卢清华

2016 年 5 月 25 日

基于 Docker 的订水管理系统开发运维

摘　要

本文主要介绍了基于 Docker 的 Web 应用的开发、部署和运维。介绍了 Web 项目基于 Spring MVC 框架设计与实现，数据库的表结构设计，Docker 基础的使用方法以及将 Web 应用部署到 Docker 中的实现方法。

自 2013 年 Docker 诞生到现在，在行业中受到越来越多的关注。Docker 提供了一个集成的技术套件，使开发和运维团队可以随时随地方便的构建、迁移、运行应用程序。Docker 容器包含应用程序软件所需要运行的一切完整的文件系统：代码，运行环境，系统工具，系统依赖库。这样就保证了它不管在什么环境下运行都会有相同的运行环境。鉴于 Docker 的优势，本文将会从 Web 应用的开发，到在 Docker 中部署、运维的过程中，来加深对 Docker 的了解，切身体会 Docker 给开发、运维人员带来的便利。

关键词：Docker；Web；DevOps；Spring MVC

Based on Docker's Order Management System operation and maintenance of water

Abstract

This paper describes the development of Web-based applications Docker, developments, deployments and operations. Introduced the Web project based on Spring MVC framework and implementation and the schema of the database table, Docker-based tutorial and deploy Web applications to the environment built by Docker.

Since 2013 Docker birth to now, more and more attention in the industry. Docker provides an integrated technology suite that enables development and IT operations teams to build, ship, and run distributed applications anywhere. Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, and system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in. Given Docker advantage above, this paper will be from the development of the Web application to be deployed, operated in Docker, and deepen understanding, get the convenience of development, operation and maintenance with Docker.

Keywords：Docker；Web；DevOps；Spring MVC

目　录

第 1 章　引言

1.1 课题背景

Docker 时代的到来。自 2013 年 Docker 诞生至今，一直践行“Build, Ship and Run Any App, Anywhere”的理念。Docker 简化并加快应用程序的开发、测试、部署、运维，促进最佳实践并催生新一代以应用为基础的微服务(MicroServices)机制，由于使用统一配置的镜像环境，可以实现快速不同环境之间的迁移。Docker 的出现无疑给人们构建 PaaS 应用程序的思维方式注入了新鲜活力。

Docker 正在迅速改变云计算领域的运作规则，并彻底颠覆云技术的发展前景。从持续集成/持续交付、微服务、开源协作以及 DevOps，Docker 一路走来已经给软件开发生命周期以及云技术实践带来了巨大变革。每天都有成千上万名新的开发者兴高采烈地参与到原有应用重构或者全新 Docker 应用开发的相关工作中来。而了解 Docker 之火能够快速呈现燎原之势的理由，正是在这个不断变化的技术世界中保持竞争优势的关键所在。[1]

1.2 课题内容

鉴于 Docker 对软件工程领域带来的巨大变革，一方面是跟上时代的步伐，体验 Docker 所带来的优势，另一方面了解如何做到基于 Docker 开发运维。本课题为基于 Docker 的订水管理系统开发运维。

该订水管理系统的设计流程以及实现用到的相关技术：

1）基于 JSP 借助 Ajax ，JQuery，使用 Spring MVC 基本完成整个系统页面设计；

2）使用 Hibernate Validator + JQuery 完成用户登录/注册、在线下单等功能以及使用 ORM 映射工具（Hibernate），使用 C3P0 数据源配置，完成数据模型的 CURD 操作；

3）为了方便调试，借助 Spring AOP，配置 slf4j 配置日志系统。

4）完成 Web 开发，打包远程部署在 ECS。

5）安装配置 Docker 运行环境

6）将该系统功能模块拆分为登录/注册模块，订单模块，日志模块，数据持久化模块等，将各个模块分别部署在不容的容器中，实现容器互通，整个项目的运行。

7）最后实现在同主机模式下各个容器的监控。

第 2 章　相关技术介绍

2.1 Spring MVC 框架简介

Spring MVC 是 SpringFrameWork 的后续产品，已经融合在 Spring Web Flow 里面。Spring 框架提供了快速构建 Web 应用的 MVC 模块。使用 Spring 插件式的 MVC 架构，可以选择是使用 Spring MVC 自带的 Web 框架，也可以选择经典的 Struts(2) Web 框架。通过相应的策略配置，Spring MVC 是高度可配置的，而且支持多种视图技术，例如 Java Server Pages（JSP）技术、Velocity、Tiles、iText 和 POI[2]。Spring MVC 内部不关心具体上层使用的是何种视图技术，这样用户就可以完全自主选择视图。Spring MVC 分离了模型对象（M）、视图（V）、分派及处理对象（C），这种分离更符合用户的需求，使它们能够更好的按需配置。Spring web MVC 和 Struts2 都属于表现层的框架,它是 Spring 框架的一部分。相对 SSH2（Spring Struts2 Hibernate）,Spring MVC 更接近原生的 Servlet 控制器的单例模式，减少了大量的过滤器的使用，从这方面而言 Spring MVC 有着更好的灵活性、性能，这一点可以从 Spring 的整体结构中看得出来，如图 2-1 所示:

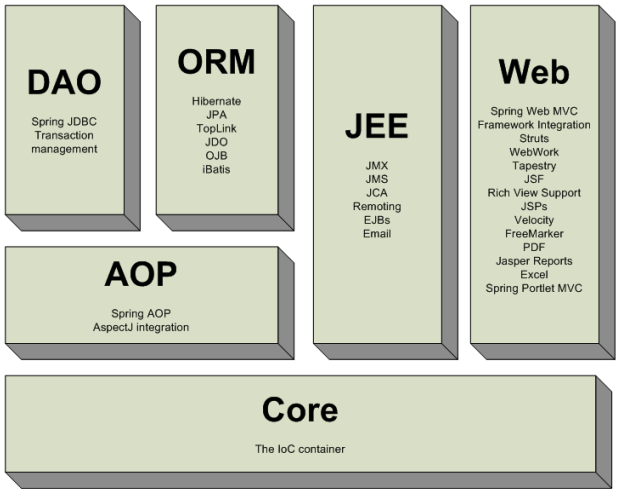

图 2-1. Spring 整体结构

Spring MVC 框架是一个 MVC 框架，通过实现 Model-View-Controller 模式来很好地将业务模型、业务处理逻辑与视图展现进行分离。从这样一个角度来说，Spring MVC

和 Struts、Struts2 非常类似。Spring MVC 的核心是分发器（DispatcherServlet），DispatcherServlet 负责将请求派发到特定的 handler。通过可配置的处理器映射（HandlerMappings）、视图解析（ViewResolution）、本地化（Locale 以及主题解析（ThemeResolution）来处理请求并且转到对应的视图，上述流程转换为下图 2-2 所示，

图 2-2.Spring MVC 处理流程图

2.2 Docker

2.2.1 Docker 背景

2013 年初，Docker 在 dotCloud —— 一个平台即服务的、以云计算为中心的公司，以一个开源项目的形式诞生。Docker 是该公司已经开发的用来在数千台服务器上运行云业务的一个自然扩展技术[3]。它是使用 Go 语言编写的，快速发展了 6 到 9 个月，加入了 Linux 基金会，将公司名改为 Docker，并且宣布将工作重心转移到 Docker 容器及其生态系统的开发[2]。随着 Docker 公司发布了第一个版本容器的产品部署以及非常活跃的社区助推，图 2-4 显示 Google Trends 近年来的关注趋势，截至本文 2016 年 6 月初，GitHub 中 Docker.git star 高达 31000+，fork 9000+，相信未来 Docker 的发展会持续高歌猛进。

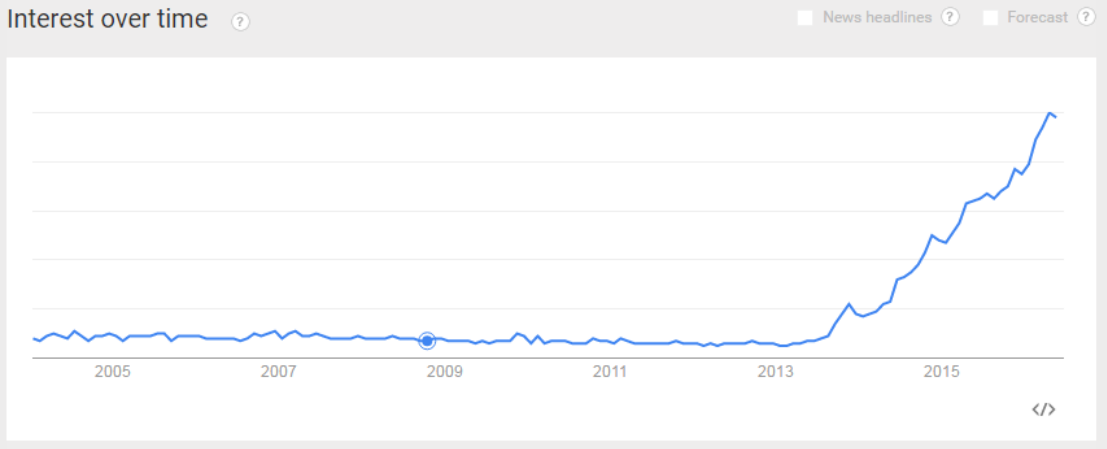

图 2-3. 近年来 Docker 在被搜索的趋势线

图 2-4. Docker 在 GitHub 中的关注度

2.2.2 什么是 Docker

Docker 是开源的容器虚拟化平台，是 PaaS（Platform as a Service）提供商 dotCloud 开源的一个基于 LXC 的高级容器引擎，源代码托管在 GitHub 上，基于 go 语言并遵从 Apache2.0 协议开源。

容器化技术并不是全新的技术，早在 1982 年，Unix 内建的 chroot 机制就是一种容器化技术，Docker 采用了 UFS（Another/Alternative/Advance Union File System 把不同物理位置的目录合并 mount 到同一个目录中）文件系统来设计一个可以层层堆叠的容器镜像文件，将容器内的所有程序都打包进 Docker 镜像中，同时支持使用一个 Dockerfile 配置文件来记录建立容器过程。只要在任何支持 Docker 平台的环境中，就可以从这个镜像来构建出一个完全一样的容器环境来执行同一个应用程序。如此一来，就可以通过 Docker 镜像，或甚至只需要 Dockerfile，就能将应用程序以及它的执行环境带走，迁移到任何支持 Docker 的环境中。通过 Docker 公司发布的 API，使用相关指令来管理容器，这样以来就等同于将容器运用标准化了[4]。

Docker 不止是一个轻量级的容器，恰当的说是一个程序开发、测试、部署的开放平台，Docker 的使用能够使你快速地交付应用。在 Docker 中，可以将应用程序分为不同的基础部分，对于每一个部分都可以当作一个应用程序来维护。Docker 能

够帮助你快速地开发、测试、部署并交付应用，并且缩短你从开发到运维应用的周期。Docker 允许软件开发者将应用以及所有的依赖打包到一个标准化单元中，这个标准化的单元就是 Docker 容器。

2.2.3 Docker 架构简介

1) Docker 架构

Docker 是 Client/Server 的架构，如图 3 所示，Docker 客户端与服务端的 Docker daemon 通过 RESTful API 进行 socket 进行交互，daemon 负责构建、运行和发布 Docker 容器。Client 和 Server 可以运行在同一个系统中，也可以部分在不同系统中，连接远程的 daemon，如下图 2-5 所示

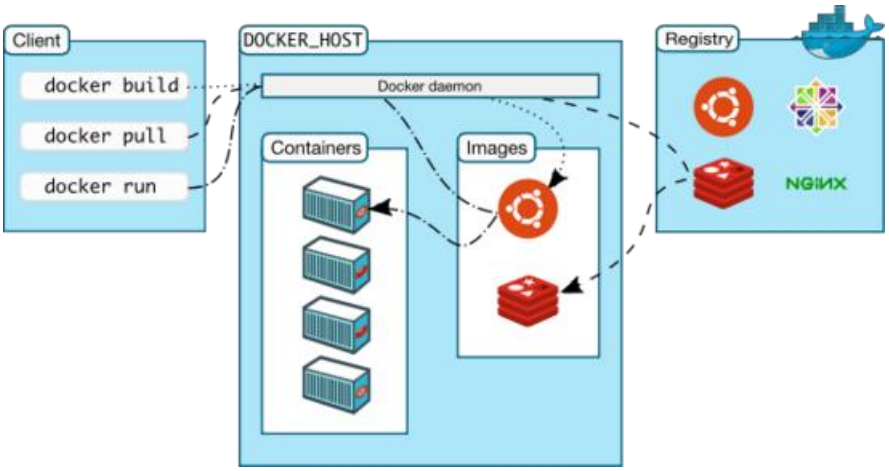

图 2-5. Docker C/S 结构图

与传统虚拟化技术相比，Docker 并没有 Hypervisor 层。因为 Docker 的虚拟化技术是基于内核的 CGroup 和 Namespace 等技术，处理逻辑与内核深度融合， 所以在很多方面， 它的性能与物理机非常接近，而且可以快速地创建、开启、停止或删除。在通信上， Docker 并不会直接与内核交互， 它是通过一个更底层的工具 Libcontainer（自 Docker1.9 之后，runC 代替了 Libcontainer）与内核交互的[5]。

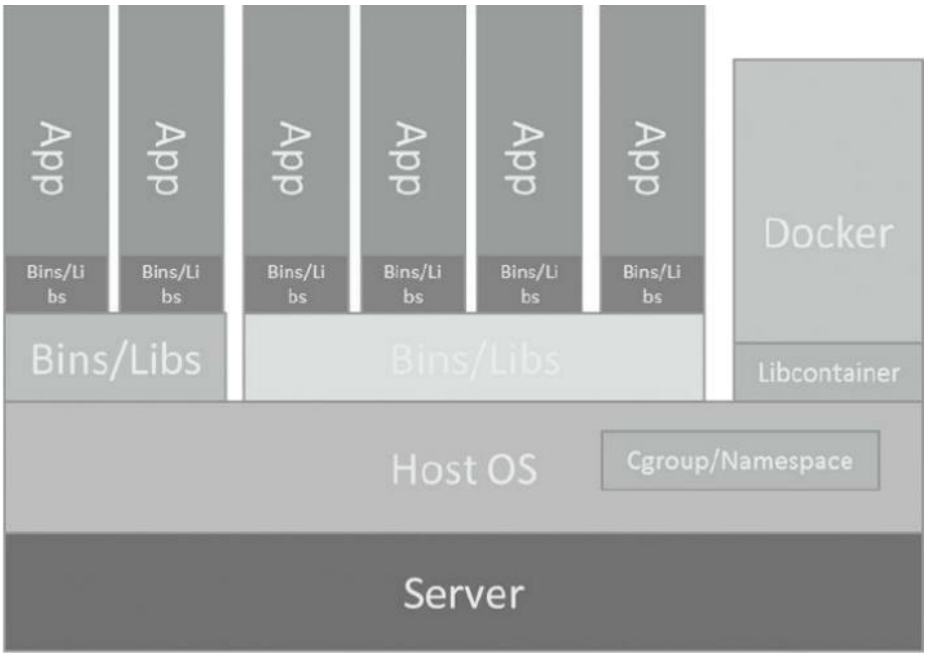

图 2-6.Docker 架构图

2) Docker 内部结构

在 Docker 内部主要包含以下三个部分

Docker 镜像

Docker 仓库

Docker 容器

(1) Docker 镜像

Docker 镜像是 Docker 容器运行时的只读模板，每一个镜像由一系列的层 (layers) 组成，如图 2-7 所示。Docker 使用 AUFS 来将这些层联合到单独的镜像中。AUFS 允许独立文件系统中的文件和文件夹(称之为分支)被透明覆盖，形成一个单独连贯的文件系统。正因为有了这些层的存在，给 Docker 在性能上有了极大的提高。当你改变了一个 Docker 镜像，比如升级到某个程序到新的版本，就会有一个新的层会被创建。因此，不用替换整个原先的镜像或者重新建立，只是一个新的层被添加或升级了。现在你不用重新发布整个镜像，只需要升级，然后将本次升级 commit，层模式使得分发 Docker 镜像变得简单和快速。

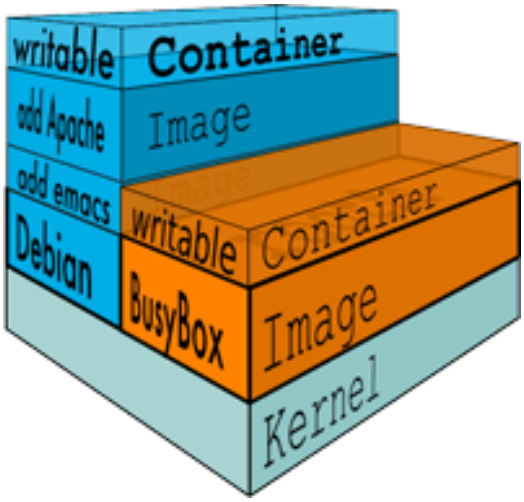

图 2-7.Docker 的层级结构

(2) Docker 仓库

用来保存镜像，可以理解为代码控制中的代码仓库,类似 GitHub，与之相似，Docker 仓库也有公有和私有的概念。

(3) Docker 容器

它是 Docker 的核心，和文件夹很类似，一个 Docker 容器包含了所有的某个应用的运行环境。每一个容器都是从 Docker 镜像创建而来的。Docker 容器可以被运行、开始、停止、移动和删除。

3) Docker 与 VM 的区别

上面提到 Docker 没有 Hypervisor 层。另外 VM 强依赖 Guest OS，也就是在使用 VM 之前必须要安装 Guest OS，如图 2-8 所示。

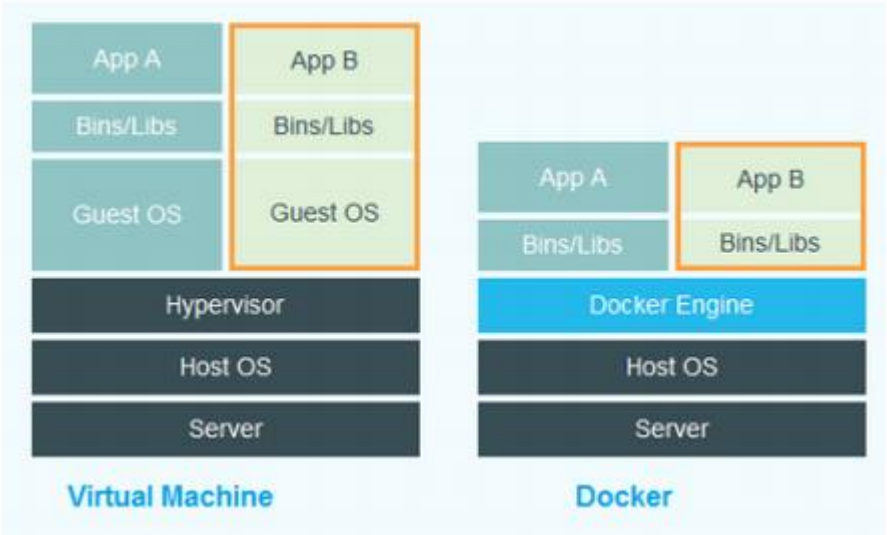

图 2-8 . Docker 与 VM 的区别

Docker 与 VM 深层次的对比，如图 2-9 所示，

对比项	Docker	对比结果	虚拟化
快速创建、删除	启动应用	>>	启动Guest OS+启动应用
交付、部署	容器镜像	==	虚拟机镜像
密度	单Node 100~1000	>>	但Node 10~100
更新管理	迭代式更新，修改Dockerfile，对增量内容进行分发、存储、传输、节点启动和恢复迅速	>>	向虚拟机推送安装、升级应用软件补丁包
Windows的支持	不支持	<<	支持
稳定性	每月更新一个版本	<<	KVM，Xen，VMware都已很稳定
安全性	Docker具有宿主机root权限	<<	硬件隔离：Guest OS运行在非根模式
监控成熟度	还在发展过程中	<<	Host，Hypervisor，VM的监控工具在生产环境已使用多年
高可用性	通过业务本身的高可用性来保证	<<	武器库很丰富：快照，克隆，HA，动态迁移，异地容灾，异地双活
管理平台成熟度	以k8s为代表，还在快速发展过程中	<<	以OpenStack，vCenter，汉柏OPV-Suite为代表，已经在生产环境使用多年

图 2-9. Docker 与 VM 深层次对比[6]

2.3.4 Docker 解决的问题

1) 现今应用部署的挑战

随着云技术的发展，服务器应用的部署已经变得越来越复杂了，以往那种通过执行脚本文件安装服务应用的方式日益突出其存在的弊端，例如对已安装软件、正在运行的服务、特定的操作系统的依赖以及特别是对系统资源的使用等方面。在软件开发过程中，使用到开发环境、测试环境、预发环境、线上环境等，环境的不一致导致问题排查困难，产品上线推迟等问题，显然这种方式已经不能跟上日益庞大的服务器应用的需求。特别是应用的隔离性、安全性、资源控制、应用迁移升级备份等方面都是对传统应用部署管理的重大挑战。

2) Docker 的解决方案

Docker 从文件系统以及网络级别使用 CGroup、Namespace、rootfs 对应用进行了深层次的资源控制、访问隔离以及文件系统隔离，再加上 Docker 容器引擎加以对容器生命周期的控制。Docker 镜像允许用户自定义方式或则使用 Dockerfile 的方式对 Docker 容器进行变更，发生变更之后，可以提交到原始镜像或者生成新的镜像。在多种环境的应用程序开发过程中，可以直接将配置好的镜像在不同环境之间传输或者使用 Dockerfile 来重新构建完全一致的镜像，这样以来就保证了开发环境的一致性，在环境发生改变的时候，只需要重新构建 Dockerfile 文件。

3) Docker 的局限性

Docker 毕竟是最近发展起来的开源的容器虚拟化平台，势必存在一些不足：

(1) Docker 是基于 Linux 64bit 的，无法在 32bit 的 Linux/Windows/Mac 环境下使用。

(2) Docker 在安全性方面存在些问题，例如从容器中获取 Host 主机的 root 权限等，但这方面的问题 Docker 团队已经确定，并采取了一些控制方案。

(3) Docker Hub 访问问题，在国内访问 Docker Hub 网站会存在一些稳定性问题。但是，我们可以采用国内的一些镜像，例如阿里云、灵雀云等。

(4) 容器技术的资源管理和运维方面，容器技术本身更适于解决大规模应用场景，所以通常都是在集群基础上的部署、运维，但是目前对这一系列任务的自动化

处理尚无统一的或者标准的框架[7]。现如今发展较好的是 Kubernetes 有望将来制定统一标准。

第 3 章　Spring MVC 框架下的 Web 设计

3.1 系统结构

Spring MVC 框架是 Spring 提供的一个构建 Web 应用程序的 MVC 实现，通过实现 Model-View-Controller 模式很好的将数据对象、业务逻辑和表现展示进行分离，降低系统的耦合性如图 3-1 所示。

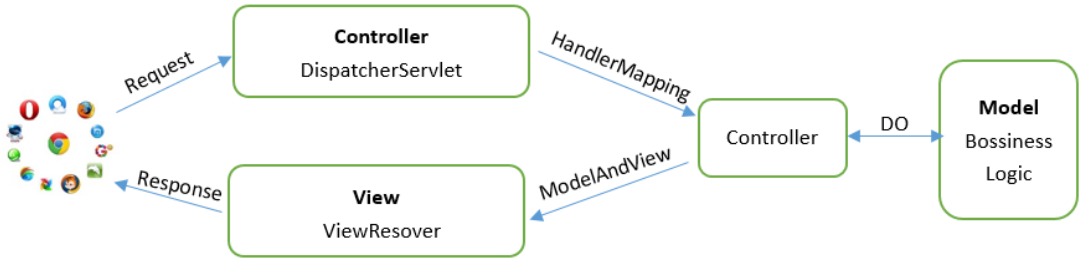

图 3-1.Spring MVC 的结构

当浏览器发起 request，Spring MVC 的核心控制器 DispatcherServlet 接收到请求，对请求做出相应的预处理，借助 HandlerMapping 返回一个适当的处理程序，同样该程序也是一个 Controller，再在该 Controller 中完成业务逻辑，并将返回的对象 ModelAndView 对象传递给 ViewResolver，通过读取 ViewResolver 的配置，将视图对象展示给用户。

本系统基于 Spring MVC 框架，将订水管理系统分为表现层、控制层、业务逻辑层、持久层四层，如图 3-2 所示

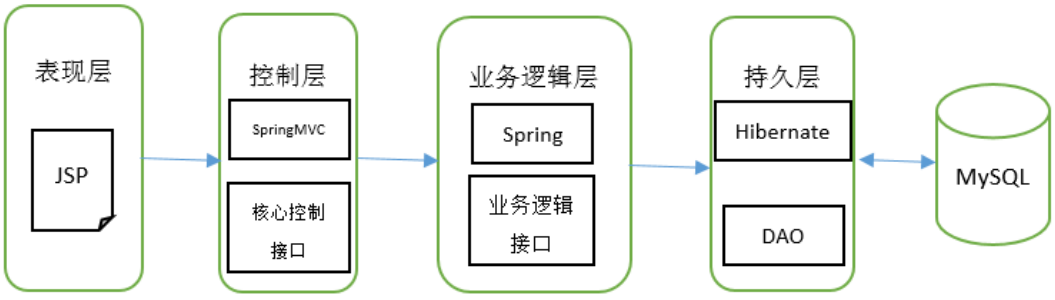

图 3-2.基于 Spring MVC 的系统分层模型

3.1.1 视图与控制器之间的接口

在众多的视图技术中，最常用的是 JSP，此外 Spring MVC 还支持 Velocity、Tiles、iText 和 POI 等。为了能够将控制器传递来的 Model 在表现层展示出来，需要对其进行相关的配置——applicationContext.xml（当然 Spring MVC 允许可以自定义配置文件的名字，而这需要在 web.xml 中加以配置），具体配置如图 3-3，

```
<!-- 核心请求分发器，截获请求，然后分发 servlet-name 中的名字来决定配置文件的名字，<serlvet-name>-servlet.xml -->
<servlet>
    <!-- servlet名字决定了配置文件的名字 -->
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!-- 加载配置文件 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:dispatcher-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

图 3-3.Spring MVC 加载配置文件的配置

在 dispatcher-servlet.xml 中使用 bean 配置的方式添加表现层视图解析器的配置，如图 3-4 所示，

```
    <!--     ========================================
             =                配置视图解析器
             ========================================  -->
  <bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <!-- 请求的前缀 -->
      <property name="prefix" value="/WEB-INF/views/" />
      <!-- 后缀 -->
      <property name="suffix" value=".jsp" />
  </bean>
```

图 3-4.视图解析器的配置

通过将控制器返回的 URL 的后缀匹配的方式，将会在/WEB-INF/views/文件夹下找到相应的 JSP 文件，如上配置，后缀是不用匹配的，默认为 JSP。由于 spring 框架是一个轻量级，无入侵性的框架实现方式，MVC 实现的可扩展性很强、可以很容易地将你选择的 Web 框架和 Spring 结合起来。只要通过 Spring 的 ContextLoadListener 启动一个 Spring 的根应用上下文，并且通过它的 ServletContext 属性在 Struts 或 WebWork 的 Action 中访问[8]。

3.2 服务器端设计

3.2.1 系统功能结构图

系统主要功能结构如图 3-5 所示，

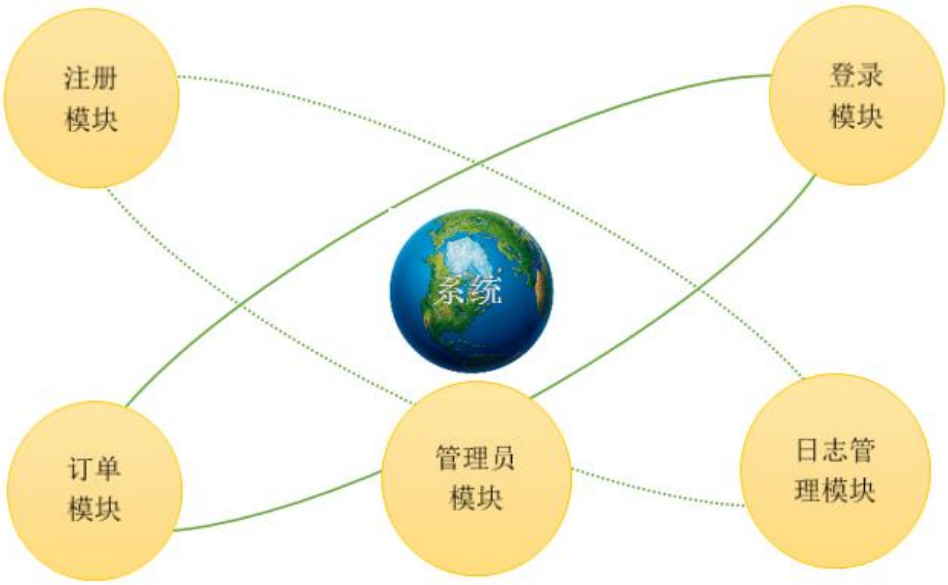

图 3-5.系统功能结构

在用户发起登录/注册服务的时候，使用 Hibernate Validator 和 JQuery 方式完成对表单的数据验证，服务器使用 MySQL 作为存储介质，并通过 Hibernate ORM 完成对数据 CURD 操作。服务器端完成的任务是提供浏览器访问接口和实现相关业务逻辑处理。服务器端的业务逻辑的包组织结构如图 3-6 所示：

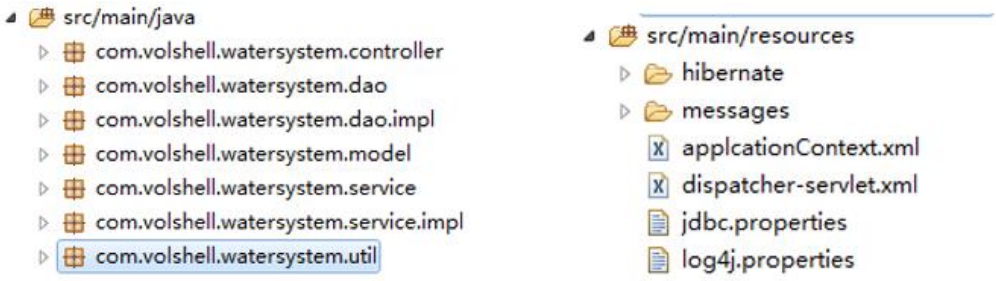

图 3-6.系统包组织结构

各个包的功能如下：

1)　com.volshell.watersystem.controller　系统核心控制器

2) com.volshell.watersystem.dao　系统数据访问对象接口

3) com.volshell.watersystem.dao.impl 数据访问对象的实现类

4) com.volshell.watersystem.model　系统业务模型

5) com.volshell.watersystem.service 系统提供的服务接口

6) com.volshell.watersystem.service.impl　服务接口实现类

7) com.volshell.watersystem.util　整个系统的工具类

3.2.2 系统的组织管理

本系统是用 maven 构建，并对相应依赖包的管理，本系统中所依赖的包，如下图 3-7 所示。同时使用 GitHub 对系统代码进行托管，托管地址为 https://github.com/smartvolshell/WaterSystem。

```
[INFO] --- maven-dependency-plugin:2.8:tree (default-cli) @ BookWater ---
[INFO] com.volshell:BookWater:war:0.0.1-SNAPSHOT
[INFO] +- org.springframework:spring-core:jar:4.2.0.RELEASE:compile
[INFO] |  \- commons-logging:commons-logging:jar:1.2:compile
[INFO] +- org.springframework:spring-webmvc:jar:4.2.0.RELEASE:compile
[INFO] |  +- org.springframework:spring-expression:jar:4.2.0.RELEASE:compile
[INFO] |  \- org.springframework:spring-web:jar:4.2.0.RELEASE:compile
[INFO] +- org.springframework:spring-beans:jar:4.2.0.RELEASE:compile
[INFO] +- org.springframework:spring-context:jar:4.2.0.RELEASE:compile
[INFO] +- org.springframework:spring-aop:jar:4.2.0.RELEASE:compile
[INFO] |  \- aopalliance:aopalliance:jar:1.0:compile
[INFO] +- org.springframework:spring-tx:jar:4.2.0.RELEASE:compile
[INFO] +- org.springframework:spring-jdbc:jar:4.2.0.RELEASE:compile
[INFO] +- org.springframework:spring-orm:jar:4.2.0.RELEASE:compile
[INFO] +- org.springframework:spring-test:jar:4.2.0.RELEASE:compile
[INFO] +- com.mchange:c3p0:jar:0.9.5:compile
[INFO] |  \- com.mchange:mchange-commons-java:jar:0.2.9:compile
[INFO] +- mysql:mysql-connector-java:jar:5.1.35:compile
[INFO] +- junit:junit:jar:4.8.1:test
[INFO] +- javax.servlet:jstl:jar:1.2:compile
[INFO] +- taglibs:standard:jar:1.1.2:compile
[INFO] +- org.apache.logging.log4j:log4j-core:jar:2.5:compile
[INFO] |  \- org.apache.logging.log4j:log4j-api:jar:2.5:compile
[INFO] +- org.slf4j:slf4j-api:jar:1.7.12:compile
[INFO] +- org.slf4j:slf4j-log4j12:jar:1.7.12:compile
[INFO] |  \- log4j:log4j:jar:1.2.17:compile
[INFO] +- org.hibernate:hibernate-validator:jar:5.2.1.Final:compile
[INFO] |  +- javax.validation:validation-api:jar:1.1.0.Final:compile
[INFO] |  +- org.jboss.logging:jboss-logging:jar:3.2.1.Final:compile
[INFO] |  \- com.fasterxml:classmate:jar:1.1.0:compile
[INFO] \- org.hibernate:hibernate-core:jar:5.0.0.Final:compile
[INFO]    +- org.apache.geronimo.specs:geronimo-jta_1.1_spec:jar:1.1.1:compile
[INFO]    +- org.hibernate.javax.persistence:hibernate-jpa-2.1-api:jar:1.0.0.Final:compile
[INFO]    +- org.javassist:javassist:jar:3.18.1-GA:compile
[INFO]    +- antlr:antlr:jar:2.7.7:compile
[INFO]    +- org.jboss:jandex:jar:1.2.2.Final:compile
[INFO]    +- dom4j:dom4j:jar:1.6.1:compile
[INFO]    |  \- xml-apis:xml-apis:jar:1.0.b2:compile
[INFO]    \- org.hibernate.common:hibernate-commons-annotations:jar:5.0.0.Final:compile
```

图 3-7.系统依赖包结构

3.2.3 业务逻辑层的设计和实现

在系统逻辑结构中，业务逻辑层位于表现层之下，数据持久层之上，依赖特定的业务规则来实现业务的处理，通过向控制层对外开放接口来实现相关服务，同时调用持久层的数据处理逻辑完成数据的持久化操作。综上可用下图 3-8 来表示，

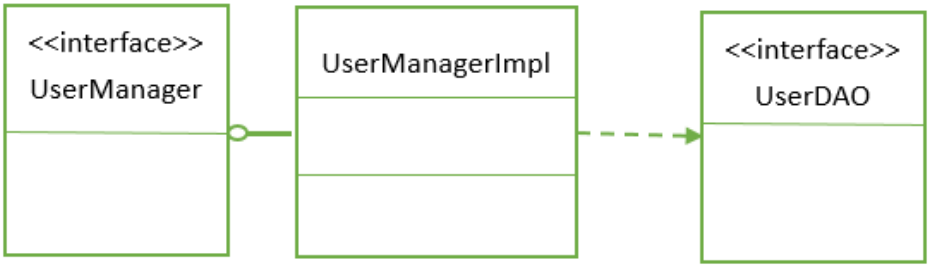

图 3-8. 业务逻辑层组件之间的关系

3.2.4 数据持久层的设计和实现

数据持久层通过 DAO 向业务逻辑层提供服务接口，DAO 的实现包括配置数据源（本系统使用 C3P0 数据源配置）、事务管理，构建 BaseDAO 基类，在基类的实现类中使用 Hibernate @Entity 注解实现 ORM，进而实现 CURD 等数据访问操作。本系统中数据持久层之间的关系如下图 3-9 所示，

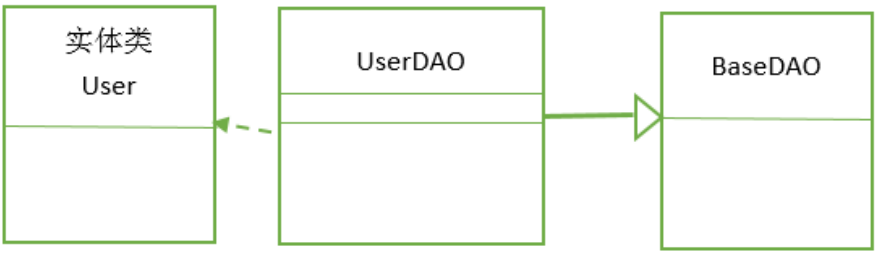

图 3-9. 数据持久层组件之间的关系

第 4 章　数据库设计

4.1 数据库表结构

本系统中涉及到表有五张，分别为用户信息表，用户订单地址表，水品表，用户订单表和订单条目表，具体表结构如下图 4-1：

#	Field	Type	Null	Key	Default	Extra
1	u_id	int(11)	NO	PRI	NULL	auto_increment
2	u_pass	varchar(25)	YES		NULL	
3	u_tel	varchar(11)	YES		NULL	
4	u_email	varchar(20)	YES		NULL	
5	u_name	varchar(20)	YES		NULL	

图 4-1.用户信息表

#	Field	Type	Null	Key	Default	Extra
1	a_id	int(11)	NO	PRI	NULL	auto_increment
2	city	varchar(255)	YES		NULL	
3	county	varchar(255)	YES		NULL	
4	h_number	int(11)	YES		NULL	
5	province	varchar(255)	YES		NULL	
6	town	varchar(255)	YES		NULL	
7	village	varchar(255)	YES		NULL	

图 4-2.地址信息表

#	Field	Type	Null	Key	Default	Extra
1	w_id	int(11)	NO	PRI	NULL	auto_increment
2	w_desc	varchar(255)	YES		NULL	
3	w_factory	varchar(255)	YES		NULL	
4	w_name	varchar(255)	YES		NULL	
5	w_price	float	YES		NULL	

图 4-3.水品详情表

#	Field	Type	Null	Key	Default	Extra
1	o_id	int(11)	NO	PRI	NULL	auto_increment
2	o_address	int(11)	NO	MUL	NULL	
3	o_date	datetime	YES		NULL	
4	status	int(11)	YES		NULL	
5	o_user	int(11)	YES	MUL	NULL	

图 4-4.用户订单表

#	Field	Type	Null	Key	Default	Extra
1	s_id	int(11)	NO	PRI	NULL	auto_increment
2	s_count	int(11)	YES		NULL	
3	o_id	int(11)	NO	MUL	NULL	
4	w_id	int(11)	NO	MUL	NULL	

图 4-5.订单条目表

4.2 系统中的表之间的关系

系统中表之间的关联关系如图 6 所示，

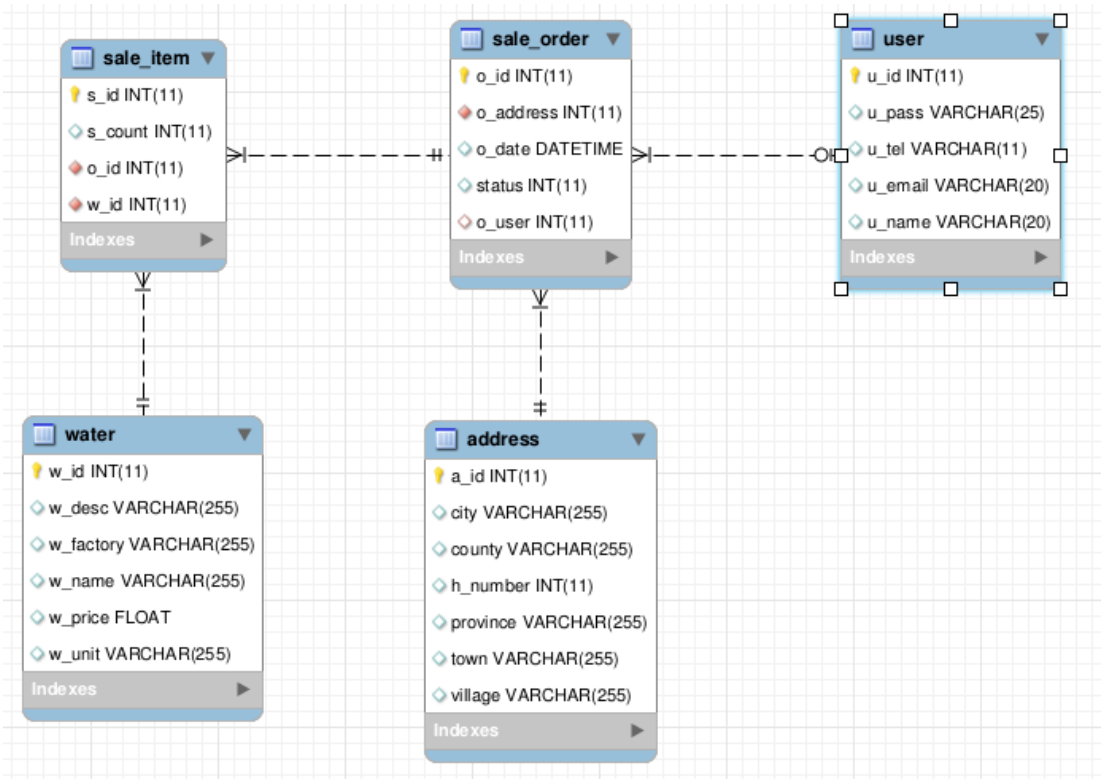

图 4-6.系统中的表结构关系

第 5 章　Docker 的使用

5.1 Docker 安装

本题目是在阿里云（CentOS 7.0）平台下进行的。由于 CentOS 7.0 库中已经带有 Docker，所以可以直接安装，如果是低于 7.0 版本，需要添加 Docker 源。

Docker 的安装与自启动

```
$ sudo yum install docker
$ sudo service docker start
$ sudo chkconfig docker on
```

获取 Docker 镜像，本题目中使用到的镜像为 Ubuntu:12.04

```
$ sudo docker pull ubuntu:12.04
Pulling repository ubuntu
ab8e2728644c: Pulling dependent layers
511136ea3c5a: Download complete
5f0ffaa9455e: Download complete
a300658979be: Download complete
904483ae0c30: Download complete
ffdaafd1ca50: Download complete
d047ae21eeaf: Download complete
```

执行上面的命令之后，会从 Docker Hub 仓库下载一个 Ubuntu:12.04 操作系统的镜像，由于在阿里云平台下面配置了阿里云的镜像，配置方式见图 5-1，所以上面会从阿里云上面获取上面版本的系统。

```
sudo cp -n /lib/systemd/system/docker.service /etc/systemd/system/docker.service
sudo sed -i "s|ExecStart=/usr/bin/docker daemon|ExecStart=/usr/bin/docker daemon --registry-
mirror=https://mvw9shcv.mirror.aliyuncs.com|g" /etc/systemd/system/docker.service
sudo systemctl daemon-reload
sudo service docker restart
```

图 5-1. 配置阿里云镜像与加速器[9]

如果需要查看我们刚刚下载的镜像，可以使用 docker images 命令来查看。

```
[root@iZ28fodz7ykZ ~]# docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
tomcat                      volshell            398258735ecb        10 days ago         536.2 MB
docker.io/centos            centos6             d487f1b804de        2 weeks ago         194.5 MB
docker.io/mysql             5.7                 b34467e98577        2 weeks ago         378.4 MB
cadvisor                    latest              ccaa711f77be        3 weeks ago         138.5 MB
ubuntu                      volshell            c633bbba2174        3 weeks ago         186.6 MB
docker.io/ubuntu            14.04               d4751aa1c40a        4 weeks ago         187.9 MB
docker.io/ubuntu            12.04               ce60cb8863fa        4 weeks ago         138.5 MB
docker.io/google/cadvisor   latest              53cdd701cc22        4 weeks ago         48.92 MB
docker.io/busybox           latest              bc744c4ab376        11 weeks ago        1.113 MB
docker.io/registry          latest              061610023430        4 months ago        422.8 MB
```

图 5-2. Docker 镜像列表

5.2 运行容器

当下载完一个镜像之后，我们就可以基于该镜像创建一个容器，创建并启动命令 sudo docker run –ti Ubuntu:12.04 /bin/bash 并通过 sudo docker ps –a 命令查看正在运行中的容器。

具体一些命令的使用，不在本文的讨论范围内，相关命令查看 Docker 文档。

5.3 Docker 数据卷

在一个应用中往往有些数据是需要在一个或者多个容器中共享，例如日志文件，Docker 提供了数据的共享机制——数据卷/数据卷容器。

5.3.1 数据卷

数据卷是能够实现一个或多个容器使用的特殊目录，它绕过 UFS，从功能上讲类似与 Linux 的 mount 命令，它提供了多种功能性的特性：

(1) 数据卷可以在容器之间共享和重用

(2) 对数据卷的修改，在容器中会立马生效

(3) 对数据卷的变更不会对镜像有影响

(4) 数据卷会一直存在，直到没有任何容器使用它

在运行一个 Docker 容器的时候，使用-v 选项，同时指定主机源目录和容器目标目录，二者使用“:”分割，多次使用-v 标记可以创建多个数据卷。

5.3.2 数据卷容器

有些时候，我们需要在多个容器之间共享数据，而且这些数据是需要持续更新的，

这种情况下，如果让每一个容器挂载一个数据卷尤为显得复杂。为了解决这个问题，Docker 提出了容器卷容器的概念。

数据卷容器其实是一个普通的容器，专门用来提供数据卷供其它容器挂载。创建数据卷容器的方式和创建普通挂载了数据卷的容器的方式是一样的，使用-v 命令。之后使用--volumes-from 命令将创建的数据卷容器挂载到其他容器中，同样多次使用该命令可以挂载多个数据卷容器。需要特别注意的是如果需要删除数据卷需要使用-v 命令删除，默认是不会删除的。除此之外，数据卷容器也为数据的热备提供方便，只需使用上面命令挂载数据卷容器同时指定 tar -cvf /backup/backup.tar /data 既可将/data 中的数据热备为 backup.rar。

5.4 Docker 网络

在正确安装完 Docker 之后，会在本机生成一个名为 Docker0 的网桥，如下图

```
[root@iZ28fodz7ykZ ~]# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.0.1  netmask 255.255.240.0  broadcast 0.0.0.0
        ether 02:42:a4:c4:c9:ca  txqueuelen 0  (Ethernet)
        RX packets 253995  bytes 357480490 (340.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 194102  bytes 52826832 (50.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 5-3.Docker0 网桥

Docker0 桥接模式网络是 Docker 提供给使用者默认的网络模式，在该种模式下面，Docker 后台程序会创建 docker0，一个虚拟的以太网桥，用于自动转发与之连接的任意网络接口间的数据包。默认情况下，该后台进程会创建一对对等接口(peer interface)，分派其一为容器的 eth0 接口，另一个则分派在主机的命名空间下，默认的名称为 vethXXX，同时从私有 IP 地址范围中为该桥接分配一个 IP 地址或子网络，由此，同一主机上的所有容器全部连接到该网桥上。如图 5-4 所示，

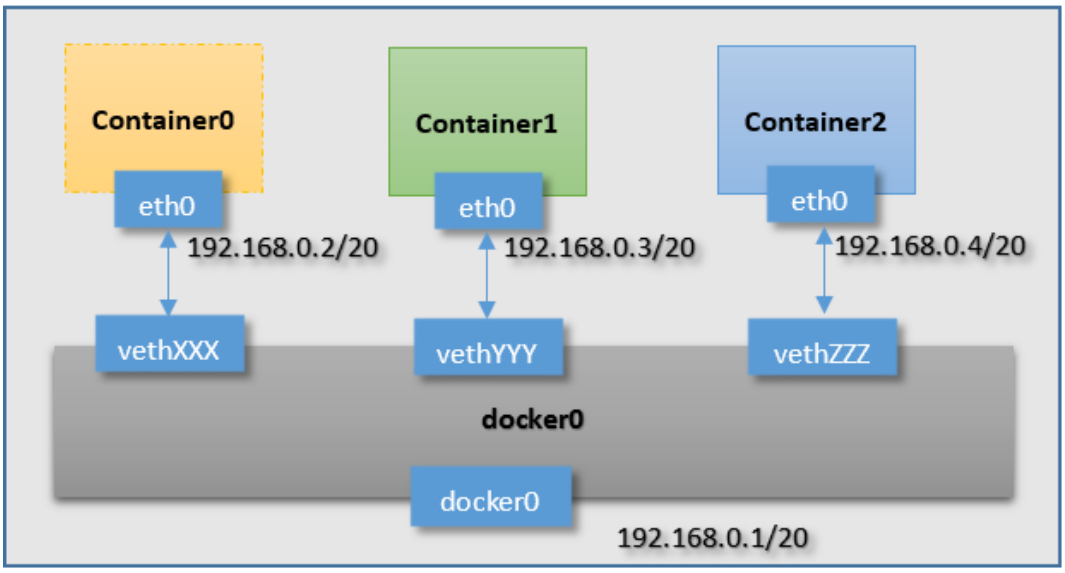

图 5-4.桥接模式网络示意图

在该种模式下可以使用-p/P 命令来指定容器对外接口。

除了上面 Docker 默认的网络模式之外，还提供其他三种网络模式：

(1) 主机网络模式，该模式下禁用容器自身网络隔离，与主机共享主机的网络命名空间。

(2) 容器网络模式，该模式下，能够重用其他正在运行的容器的网络命名空间。

(3) 无网络模式，该模式下对容器的进行配置，使用者可以自行对网络进行定义。

5.5 Docker 容器互联

Docker 容器给复杂的应用带来的生机，尤其是微服务概念的出现，开发者可以更新应用程序的单个组件，而不影响其他的部分。但是各个组件之间的交互是必需的，Docker 允许使用者使用相关命令实现勇气之间的互联。

5.5.1 同主机容器互联

通过 link 方式创建容器，然后我们可以使用被 link 容器的别名进行访问，从而解除应用对 IP 的依赖，这样会在容器的/etc/hosts 文件中生成相应的 DNS 解析。不幸的是 link 方式只能解决单机容器间的互联。跨主机情况下，容器的互联需要其他的方式。

5.5.2 跨主机容器互联

容器的跨主机互联的基本思想可以使用下图来表示：

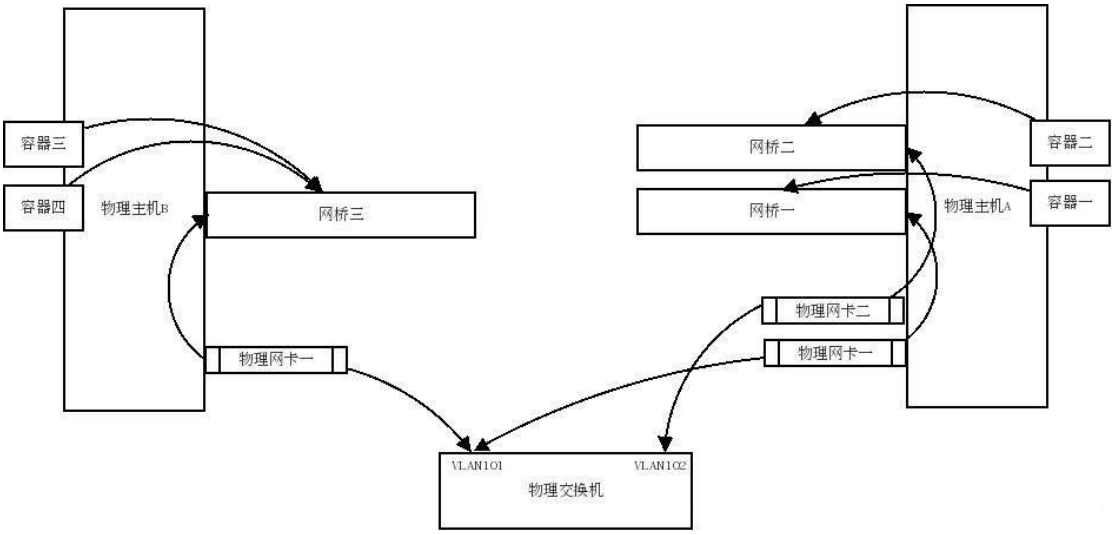

图 5-5.跨主机容器互联的原理[10]

如果要实现跨主机的容器互联可以使用下面的方式：

(1) 修改主机中的 hosts 文件的方式。

2). Docker 官方提供了一种 ambassador 的 agent 方案。需要借助 svendowideit/ambassador 的 image，将不同 host 进行解耦合。如图 5-6 所示，

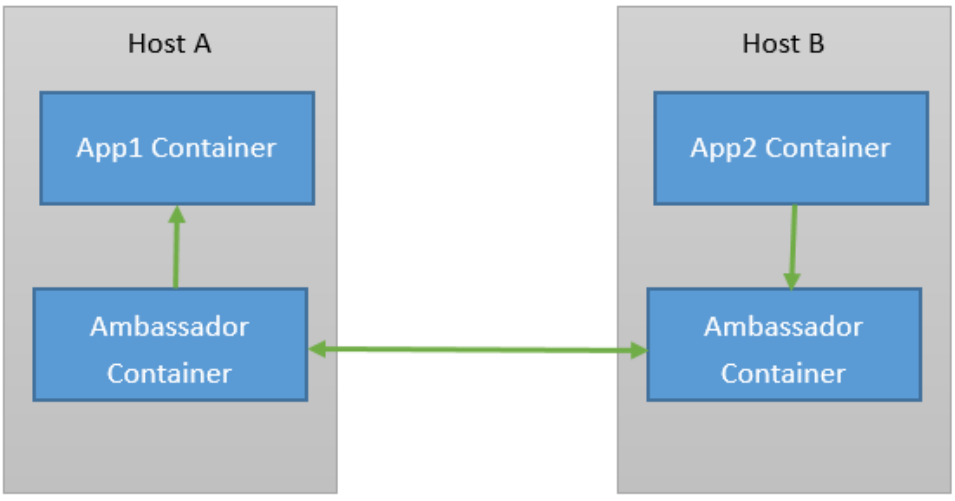

图 5-6. Ambassador 方式跨主机容器互联

5.6 Docker 容器监控与编排

5.6.1 Docker 容器监控

随着 Docker 的发展，Docker 容器中性能越来越收到重视，比如容器的 CPU，Memory 等性能指标的控制显得尤为重要，对应用的部署有着相当重要的决策作用。

本系统中的 Docker 容器监控方案选择使用 Google cAdvisor，这是一个 Docker 容器内封装的实用工具，能够搜集、集料、处理和导出运行中的容器的信息。通过它可以看到 CPU 的使用率、内存使用率、网络吞吐量以及磁盘空间利用率。cAdvisor 只能监控同一主机的容器的运行状况，对与跨主机的多节点是无效的。cAdvisor 的启动方式如下图 5-7 所示，

```
sudo docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:rw \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  google/cadvisor
```

图 5-7. cAdvisor 的启动

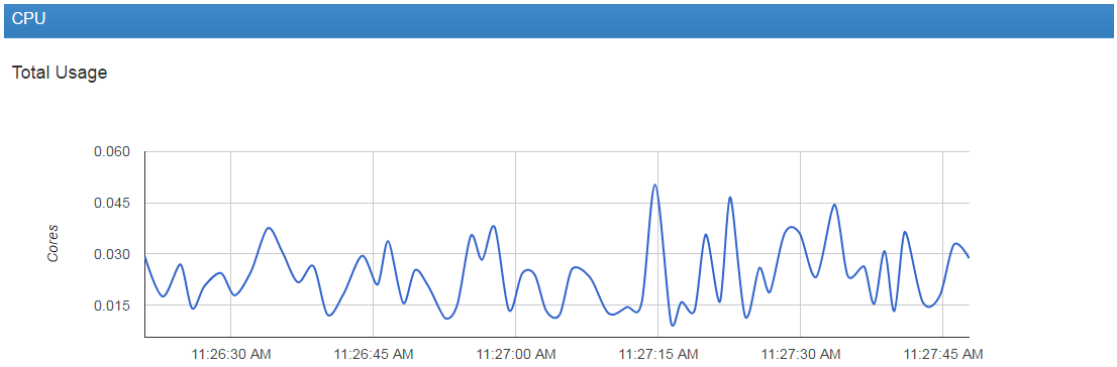

图 5-8. cAdvisor 对 CPU 的监控

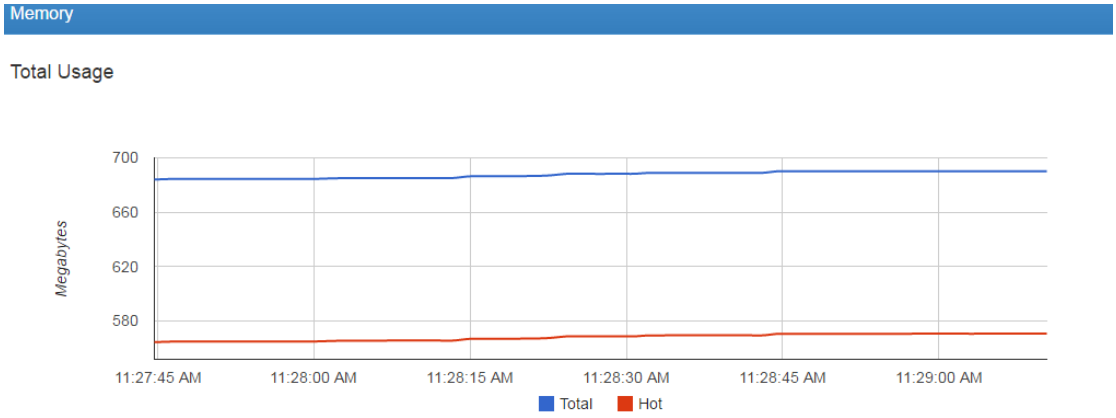

图 5-9. cAdvisor 对 Memory 的监控

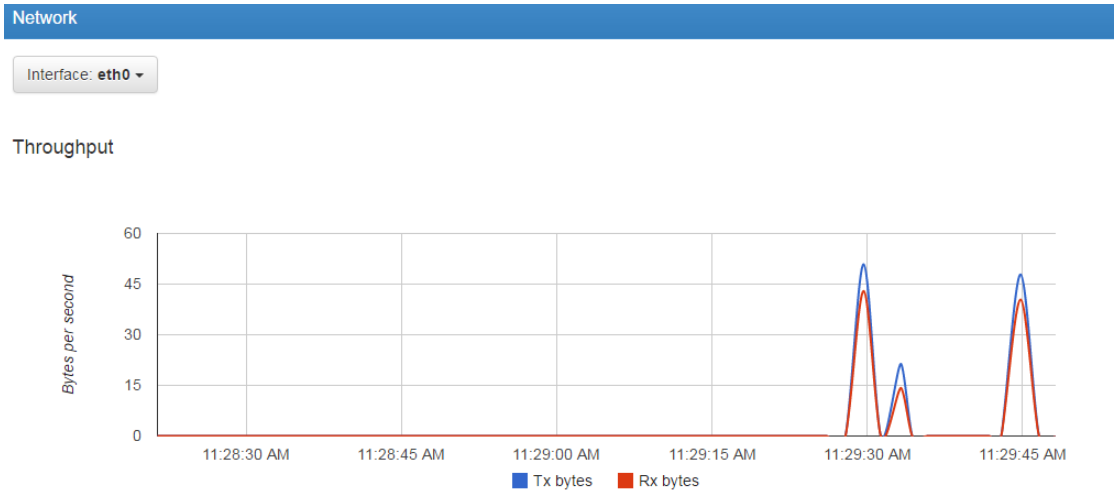

图 5-10. cAdvisor 对 Network 的监控

另外，Cloud Insight 是国内唯一一款的 Docker 监控软件，支持多种操作系统、云主机、数据库和中间件的监控，原理是在平台服务仪表盘和自定义仪表盘中，采集并处理 Metric，对数据进行聚合与分组等计算，提供曲线图、柱状图等多样化的展现形式。

5.6.1 Docker 容器编排

现在应用的编排工具不止是像 cAdvisor (Container Advisor)完成容器的监控，编排工具在监视服务器容器的同时，还要保证容器的正常运行，以及容器集群高可用性。编排的一个重要理念是利用有限可用的计算资源动态部署应用和服务，在保证程序正常运行的同时，提高资源的利用率，对于分布式系统而言，编排是一个重要的工具，完成应用的异地部署和集中管理。Docker 官方提供一些编排工具，包括 Docker Compose，Docker Machine，Docker Swarm。

1) Docker Compose

它是 Docker 官方发布的编排工具之一，主要功能是快速在集群中完成应用的部署。与 Dockerfile(见 5.7 章节)相比，管理者使用 Dockerfile 来管理一个单独的容器，而 Compose 允许使用者利用 YAML 模版文件管理一组相关的容器。

2) Docker Machine

Docker Machine 是一个简化 Docker 安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装 Docker，比如 VirtualBox、 Digital Ocean、Microsoft Azure。用户在安装了 Docker Machine 之后，只需要几个简单的命令就可以完成 Docker 的安装。

3) Docker Swarm

Swarm 是 Docker 公司在 2014 年 12 月初发布的一套较为简单的工具，用来管理 Docker 集群，它是 Docker 的原生集群管理工具。它将 Docker 集群中的宿主机抽象为一个巨大的单一虚拟主机。Swarm 借助标准的 Docker API 接口作为其前端访问入口，使用全新的调度策略实现集群中容器的调度，Docker Client 可以直接与 Swarm 通信。Google 推出的集群管理工具 Kubernetes（K8S），正逐步成为业界关注的焦点。

5.7 Dockerfile 的使用

除了使用 docker pull 命令从 Docker Hub 或其他镜像库拉去镜像之外，Docker 允许使用者自行在基础镜像的基础上制作自己需要的镜像。Dockerfile 由一行行命令语句组成，并且支持以#开头的注释行。一般的，Dockerfile 分为四部分：基础镜像信息、 维护者信息、 镜像操作指令和容器启动时执行指令。Docker 中常用指令说明见下表 1 所示，

命令名称	简介	命令名称	简介
FROM	首个命令，指定来自基础镜像	ADD/COPY	拷贝文件到容器
MAINTAINER	维护作者信息	CMD	指定容器启动后命令
RUN	指定运行的命令	ENV	设置容器环境变量
VOLUME	指定容器挂载目录	EXPOSE	指定对外端口

表 5-1.Dockerfile 命令及简介

在完成 Dockerfile 的编写之后，可以通过使用命令 docker build 来创建镜像，通过-t 参数来指定镜像标签。

第 6 章　基于 Docker 的 Web 部署

基于上面对 Docker 理解，对于开发和运维人员，最方便的莫过于只需要一次的构建或配置，就可以在任何环境中部署应用。开发者可以使用一个标准的镜像来构建一套开发、测试、发布容器，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。再加上容器能够被快速地创建，快速迭代应用程序，这样以来能够大量地节约整个软件工程过程的周期。下图 6-1 述了 在阿里云 ECS Docker 平台下，如何使开发、测试和运维之间紧密合作，各司其职， 效率更高。

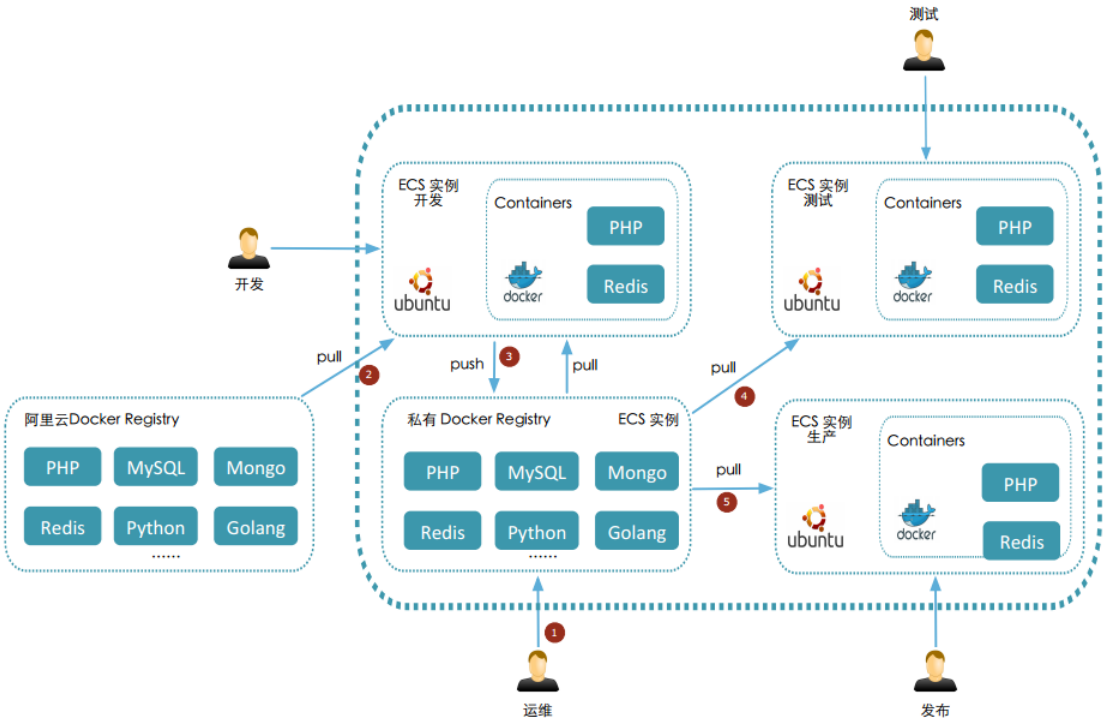

图 6-1. 阿里云 ECS Docker 环境开发、测试、运维之间的协作[11]

6.1 开始前的准备

在开始配置 Docker 环境之前需要首先按照上面的方法安装最新版本的 Docker，因为执行 Docker 相关命令需要 root 权限，为了方便输入，每次不必输入 sudo，将当前用户添加到 Docker 用户组里面，先关配置如下图 6-2 所示，

```
#!/bin/bash

#添加用户组
sudo groupadd docker
#添加当前用户到docker用户组
sudo gpasswd -a volshell docker
#重启docker服务
sudo service restart docker
#测试docker
docker version
#重启系统
sudo reboot
```

图 6-2. 添加该用户到 docker 用户组

6.2 基础镜像准备

为了使本项目中所有的容器使用一致的环境，在这之前需要配置一个基于 Ubuntu:12.04 的基础镜像，同时在该镜像中添加基础服务/软件，包括 sshd、一些网络管理工具等，具体配置如下图 3 所示，

```
FROM ubuntu:12.04
MAINTAINER volshell "lovestone1115@gmail.com"

#添加源，安装ssh
RUN echo "deb http://mirrors.163.com/ubuntu precise main universe" > /etc/apt/sources.list

RUN apt-get update && apt-get upgrade

RUN apt-get install -y openssh-server

#安装网络配置工具
RUN apt-get install -y bridge-utils inetutils-ping net-tools

#对外暴露22端口
EXPOSE 22
```

图 6-3.基础镜像的配置方案

6.2.1 open-jdk 镜像的制作

完成上面镜像的下载配置之后，提交变更为一个新的镜像，命名为 ubuntu:volshell，在此基础上启动一个容器，并安装 open-jdk，并将该次修改提交为一个镜像，为之命名为 jdk:volshell，如图 6-4 所示，

```
# Pull base image.
FROM volshell:ubuntu

# Install Java.
RUN \
  apt-get update && \
  apt-get install -yq unzip openjdk-7-jdk && \
  rm -rf /var/lib/apt/lists/*
# Define commonly used JAVA_HOME/JRE_HOME variable
ENV JAVA_HOME /usr/lib/jvm/java-7-openjdk-amd64
ENV JRE_HOME  /usr/lib/jvm/java-7-openjdk-amd64/jre
```

图 6-4. 配置 open-jdk 的 Dockerfile

之后基于该镜像创建一个容器，并完成 jdk 的环境测试。

6.2.2 tomcat 镜像的制作

在测试 jdk 环境测试成功之后，开始在此基础上创建 tomcat 镜像，并完成相关环境配置如下图 5 所示，

```
#Install tomcat7
RUN wget  http://mirror.bit.edu.cn/apache/tomcat/tomcat-7/v7.0.69/bin/apache-tomcat-7.0.69.zip && \
    unzip apache-tomcat-7.0.69.zip
RUN cp apache-tomcat-7.0.69/ -R /opt/

ADD tomcat-users.xml /opt/apache-tomcat-7.0.69/conf/

RUN chmod a+x /opt/apache-tomcat-7.0.69/bin/*.sh

#Define commonly user CATALINA_HOME CATALINA_BASE
ENV CATALINA_HOME /opt/apache-tomcat-7.0.69

# Define default command.
CMD ["/bin/bash"]
```

图 6-5.tomcat7 安装配置 Dockerfile

其中 tomcat-users.xml 的配置文件如下图 6-6 所示，

```
[root@iZ28fodz7ykZ tomcat]# cat tomcat-users.xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
<role rolename="manager-gui"/>
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admin" roles="manager,manager-gui,manager-script" />
</tomcat-users>
```

图 6-6 tomcat 配置

6.2.3 MySQL 镜像的制作

对于 MySQL 镜像的制作，同样使用上述 Dockerfile 的方式完成配置，完成配置之后，查看 MySQL 镜像情况并运行 MySQL 容器，完成相关配置测试。

6.2.4 其他配置

除此之外，配置 cAdvisor 的安装配置，如下图 6-7 配置，

```
docker run   --volume=/:/rootfs:ro\
             --volume=/var/run:/var/run:rw\
             --volume=/sys:/sys:ro\
             --volume=/var/lib/docker/:/var/lib/docker:ro \
             --publish=8080:8080   --detach=true   --name=cadvisor   google/cadvisor:latest
```

图 6-7.cAdvisor 的启动

6.3 Web 应用的远程部署

在本地完成 Web 的开发完成之后，使用 maven 将其远程部署到阿里云，在这之前在阿里云中实现 Web 环境的配置，将部署目录以 volume 挂载的方式在 Docker 容器中使用，同时将本项目的日志文件，数据库都将会以目录挂载的方式。

6.3.1 配置日志文件的数据卷容器

在本 Web 项目中日志文件的默认存储位置为/home/volshell/logs/，通过运行一个挂载该目录的数据卷容器，供 Web 项目使用，启动方式如下图 6-8 所示，

```
#1.配置日志数据卷容器
docker run -tid --name=log -v /home/volshell/logs:/home/volshell/logs/ \
        ubuntu:volshell /bin/bash
```

图 6-8.日志容器卷的启动

6.3.2 配置数据库容器

本项目中使用 MySQL5.7，具体配置如图 6-9 所示，

```
#2.配置数据库数据卷容器
docker run --name=db001 -p 3406:3306 -e MYSQL_ROOT_PASSWORD=volshell -d mysql:5.7
```

图 6-9. MySQL 容器的启动

使用 ssh 登录 MySQL，创建数据库 watersystem，当然创建数据的操作，也可以在

创建 MySQL 的 Dockerfile 中使用 RUN 命令完成数据的创建。

6.3.3 项目源代码的管理

为了方便程序代码的升级，现将程序源代码放入一个单独的容器中加以管理，配置如图 6-10 所示，

```
#3.配置web应用代码管理数据卷容器
docker run -tid  -v /var/www/BookWater:/opt/apache-tomcat-7.0.69/webapps/BookWater \
        --name=watersystem \
        ubuntu:volshell /bin/bash
```

图 6-10. 核心代码数据卷

6.3.4 启动应用

在完成上述配置之后，启动 Web 应用，启动方式如图 6-11 所示，

```
#4.启动web程序，并挂载上述数据卷容器
docker run -it -p 8088:8080 --name=myapp \
        --link=db001:db \
        --volumes-from=log \
        --volumes-from=watersystem \
         tomcat:volshell
```

图 6-11. Web 应用的启动

6.3.5 应用启动测试

测试应用是否启动，转到 http://42.96.198.24:8088/BookWater/index.jsp，如图 6-12 所示，

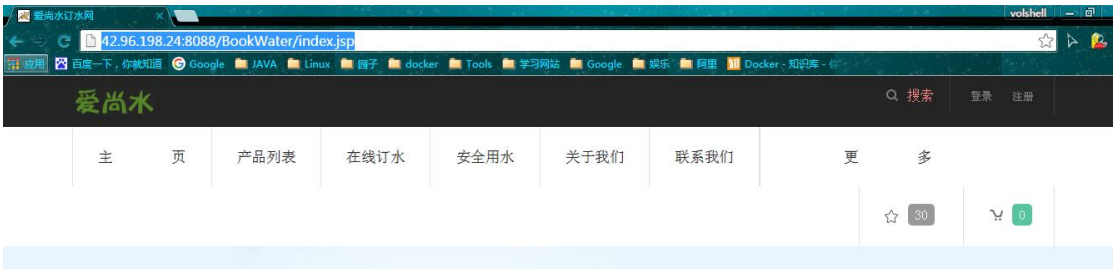

图 6-12.应用测试

6.4 Docker 性能监控

项目使用的平台环境配置如下表，

实例名称	CPU	Memory	Network
CentOS7u0_64	阿里云 单核	1024MB	1Mbp(峰值)

表 6-1.项目应用平台环境

启动 cAdvisor，转入 http://42.96.198.24:8080/containers/查看当前平台性能，

(1) CPU 占比

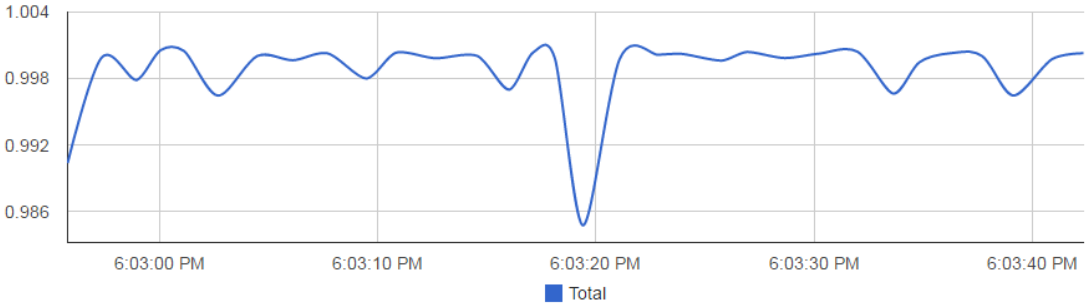

图 6-12. cAdvisor 对 CPU 的监控

(2) Memory 占比

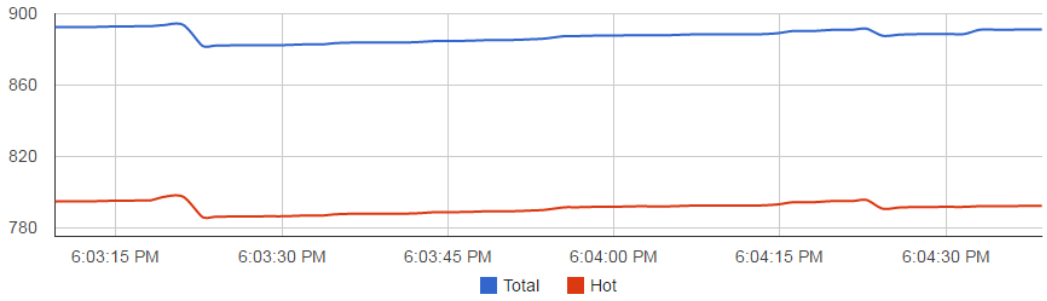

图 6-12. cAdvisor 对 Memory 的监控

(3) Network 占比

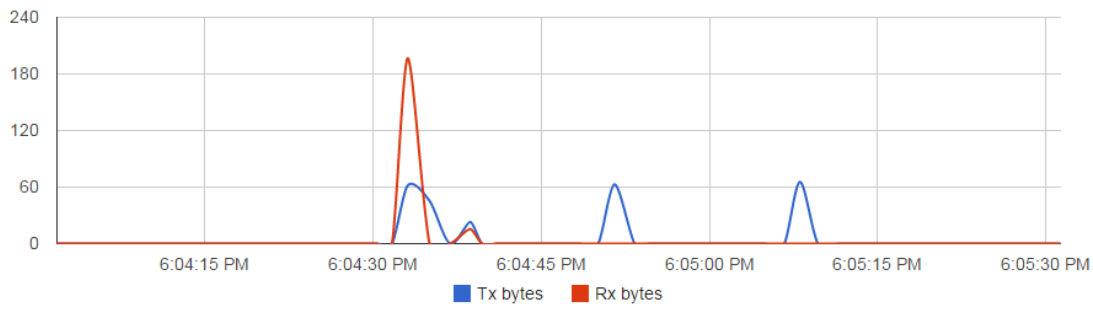

图 6-12. cAdvisor 对 Network 的监控

第 7 章　总结与展望

7.1 工作总结

本项目的构建一个 Web 应用，然后了解基于 Docker 这项新技术的部署与运维。项目初期，一方面由于签约公司的需要一心扑在 Web 方面的技术学习研究，另一方面项目初期对整个项目的理解不是很透彻，导致重心有所偏差，还好能够与指导老师卢清华老师及时交流，及时反馈不至于项目最终延期。

Docker 是一项 2013 年发展起来的新产品，几乎每个月都会有一个新版本被释放出来，这无形中给学习添加了一些困难。当面对一些问题，去网上搜些资料，总会因为版本的问题不能实质性解决问题，最后只能硬着头皮翻阅官网上面释放出来的文档。

在将 Web 应用部署到 Docker 的过程中，需要一遍一遍的测试每一个环境，而且每一次的 pull 镜像的过程，都是非常漫长的。在这种环境下，学会了如何“并发性”的工作，进而提高工作效率。

7.2 后期展望

鉴于本项目的重心是探究基于 Docker 的 Web 应用的部署运维，这个过程中存在的不足如下：

前期一直关注如何借助技术手段将 Web 做的尽可能的漂亮，痴迷技术的研究，导致中期进度稍慢，而 Web 项目功能上不是很丰满。

本项目主要是关注如何在单节点上部署运维 Web 项目，虽然学习了一些多节点 Docker 的编排技术，但是由于硬件资源的限制，没有深入在多节点加以实践。

致　谢

随着毕业设计的完成，四年的大学生活已经接近尾声，即使再有不舍，也要选择继续。四年的时光，让自己从懵懵懂懂到逐步成熟长大。而在这个过程中，离不开老师和同学们的支持帮助。四年中，自己曾全心全意地带领班级为荣誉而战，到后来为朋友们最终能够实现自己的工作、学习深造的梦想而喜悦。其实最应当感谢的是父母的无私支持，从四年前一心一意我条件支持我重新选择升学的坚决，到四年后的今天为我找到一份满意工作而喜极而泣，常怀感恩之心，感谢父母。除此之外，非常感谢女朋友在一起五年时间给我的支持和包容，由于自己做一些学生工作，会占用一些陪她的时间，而她非但没有不开心，反而鼓励我做事就要做好，得女如此，夫复何求！

在毕设项目研究学习过程中，感谢毕业设计的指导老师卢清华老师，总会在我不知所措的时候，答疑解惑；在咨询问题的时候，总会第一时间不厌其烦的耐心讲解，良师益友，大致如此吧。

参考文献

[1] Lucas Carlson. 4 ways Docker fundamentally changes application development.

http://www.infoworld.com/article/2607128/application-development/4-ways-docker-fundamentally-changes-application-development.html.2014-09-18.

[2] HeeSscia. 三大框架原理 http://blog.sina.com.cn/s/blog_606e85760101dthn.html . 2014-10-16.

[3] DockerOne. Docker 入门实战. Docker 社区 DocKOne.io 推出.2014-02-15。

[4] Adrian Mouat. Using Docker-developing and deploying software with containers. 2015-9-30.

[5] Dirk Merkel. Docker—用于统一开发和部署的轻量级 Linux 容器.

http://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment. 2014-05-19.

[6] 徐安 Docker 浅见 [http://blog.csdn.net/cxboyee/article/details/50698398] 2016-02-19

[7] 华为 Docker 实践小组 Docker 进阶与实战--容器技术系列.2016-1-5.

[8] Totobo. 使用 Struts + Spring + Hibernate 组装 Web 应用[EB/OL].

http://www.kissjava.com/doc/J2ee/websevice/2005-06-07/19451118128953.html. 2013-09-11.

[9] 阿里云 Docker 镜像配置操作手册

[https://cr.console.aliyun.com/?spm=5176.1971733.0.2.JbKBiF#/docker/booster].阿里云官网

[10] 美味小鱼. [http://blog.csdn.net/smallfish1983/article/details/38845351] CSDN 博客

[11] 阿里云 Docker 使用手册. ECS Docker 实战