# 运通汇系统高可用架构2.0

[运通汇系统高可用架构2.0 1](#_Toc7148)

[1.1. 网络区域化 2](#_Toc5389)

[1.2. 应用集群化 2](#_Toc30560)

[1.2.1. 构建公共资源集群 3](#_Toc18869)

[1.2.2. Web应用集群 4](#_Toc32636)

[1.2.3. Dubbo业务组件集群 4](#_Toc19912)

[1.2.4. Nginx应用集群 4](#_Toc29229)

[1.2.5. MySQL集群 5](#_Toc31888)

[1.3. 运营监控化 5](#_Toc18022)

[1.3.1. 日志监控 5](#_Toc32468)

[1.3.2. 虚拟机硬件监控 5](#_Toc24808)

[1.3.3. Nginx运行监控 6](#_Toc28937)

[1.3.4. Tomcat运行监控 9](#_Toc21564)

[1.3.5. Dubbo运行监控 11](#_Toc3024)

[1.3.6. Redis 运行监控 12](#_Toc11707)

[1.3.7. MQ运行监控 13](#_Toc26994)

[1.3.8. MySQL运行监控 13](#_Toc8085)

[1.3.9. FAST DFS文件服务运行监控 14](#_Toc6210)

[1.4. 升级部署平滑化 14](#_Toc6189)

[1.4.1. 基础环境升级 14](#_Toc26890)

[1.4.2. 基础应用升级 15](#_Toc6475)

[1.4.3. 业务应用升级 16](#_Toc8689)

[1.4.4. 数据库升级 16](#_Toc30195)

[1.4.5. 项目的升级 16](#_Toc16763)

[1.4.6. 紧急bug升级 16](#_Toc5094)

[1.4.7. 快速升级 16](#_Toc6799)

[1.5. 配置集中管理化 16](#_Toc30217)

[1.5.1. 单台虚拟节点系统配置 16](#_Toc6567)

[1.5.2. 基础环境配置 17](#_Toc16215)

[1.5.3. 应用容器配置 18](#_Toc17563)

[1.5.4. 业务组件服务配置 20](#_Toc6658)

[1.6. 环境镜像化 20](#_Toc20900)

[1.6.1. 虚拟机镜像 20](#_Toc3306)

[1.6.2. 单台虚拟节点系统docker镜像 21](#_Toc22406)

[1.6.3. 基础环境docker镜像 21](#_Toc22941)

[1.6.4. 业务应用docker镜像 21](#_Toc26629)

[1.7. 项目版本化 21](#_Toc3958)

## 网络区域化

不同地区、不同运营商的用户访问运通会员访问系统的速度，根据会员的地理分布和入网运营商，购买CDN代理转发服务，把对应地理分布的访问客户负载到对应的区域集群服务上（成都，北京2个集群服务）

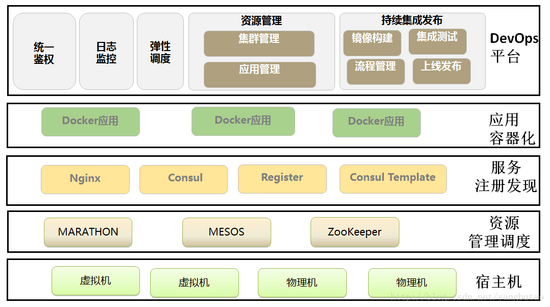
数据同步方案：

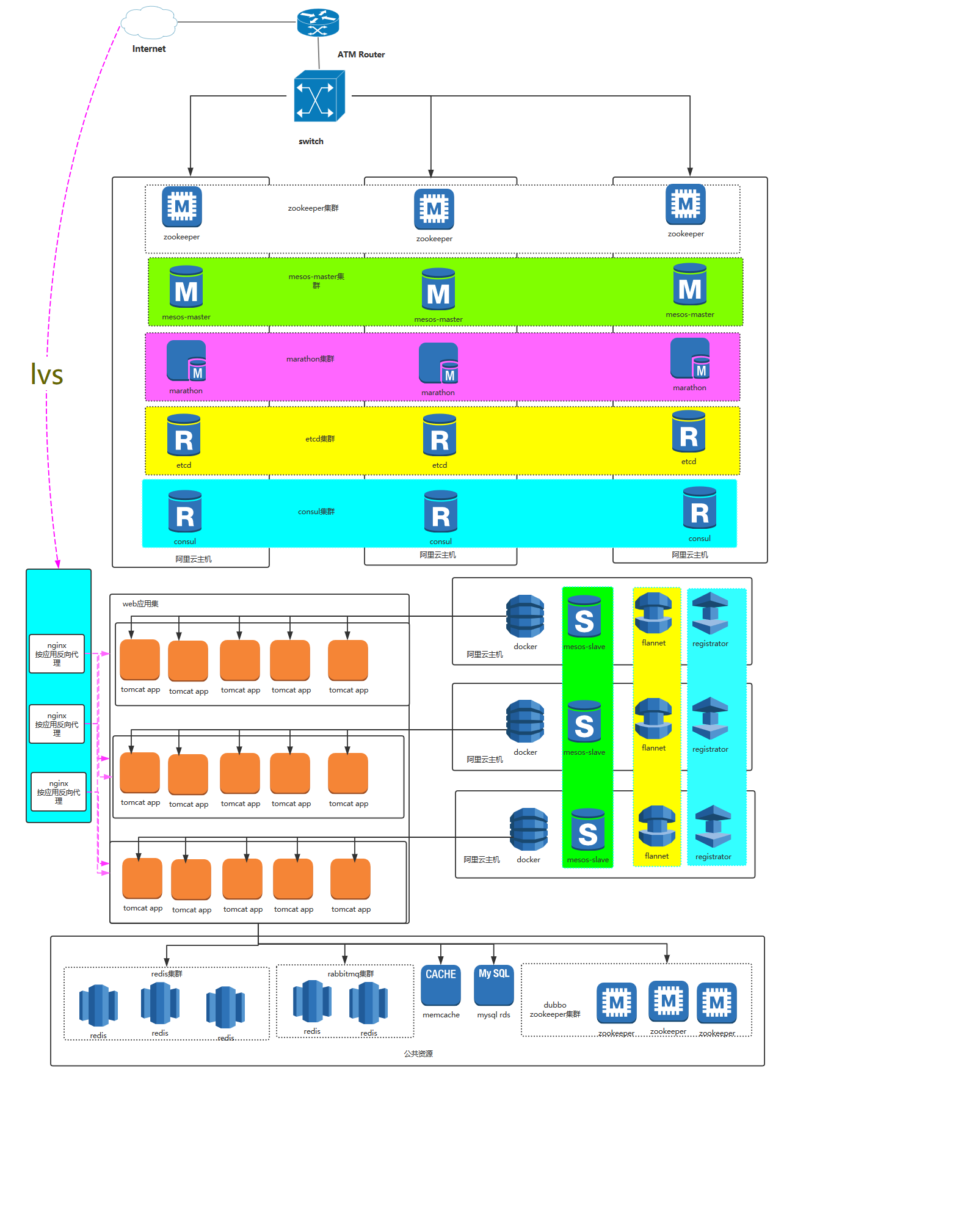
1. 成都集群和北京集群同时具备写入和读取节点，集群之间做双向同步来存储两个集群的全量数据
2. 成都集群和北京集群只有读取节点，写入节点统一从北京服务节点写入，写入后把数据同步到成都和北京集群

成都集群：

北京集群：

## 应用集群化





### 搭建集群环境

1、搭建Mesos

由 ZooKeeper、Mesos Master、Mesos Agent 构成了基础的 Mesos 数据中心操作系统，可以统一管理机房中的所有物理机，负责资源层面的调度，是二层调度系统最基础的运行环境 。

2、搭建Marathon调度架构

Marathon配合Mesos为容器提供动态的资源调度，同时提供负载平衡，横向扩展和可恢复性等基础服务能力,上层分布式应用不再是静态的拓扑结构，而是透明地运行在动态的资源环境中。

3、搭建Consul服务发现架构

Consul：包含服务发现，健康检查和 KV 存储功能的一个开源集群管理系统，marthon调度系统使用 Consul 的服务发现和健康检查机制提供基础的服务发现功能，使用 KV 存储功能来存储 集群服务的 metadata。

4、安装docker镜像环境架构

docker配合mesos、marathon快速构建应用运行环境，和项目运行环境

### 构建公共资源集群

实现方案：

为方便运通汇各个系统更快更高效的调用缓存服务(memcahed/redis)、消息队列服务(rabbitMQ)、文件上传/下载服务(fastDFS),采用mesos+marthon+docker搭建redis缓存集群、rabbitmq消息队列集群、文件服务器集群

搭建步骤：

3、安装redis镜像

4、安装rabbitMQ镜像

5、安装fastDFS镜像

调用方式：

Redis://api/dbname/tablename/key

RabbitMq://api/appname/

fastDFS://api/file(image)

### Web应用集群

实现方案：

为提高运通汇各个系统模块的web服务性能和稳定性，采用mesos+marthon+docker+Tomcat集群的方式跑所有web app

1 Tomcat cluster = 1\*HAproxy + 3\*tomcat

搭建步骤：

1. 安装mesos
2. 安装marthon
3. 创建Tomcat集群

调用方式：

HAproxy ip

### Dubbo业务组件集群

实现方案：

为Tomcat应用提供稳定的业务调用服务，搭建多台业务dubbo组件，组成集群服务

1dubbo cluster = 2\*zookeeper+3\*dubbo

搭建步骤：

1. 创建Dubbo业务组件集群

调用方式：

zookeeper://注册中心master ip:2181/api

### Nginx应用集群

静态WEB应用站点集群

搭建步骤：

1nginx cluster = 1\*HAproxy + 3\*nginx实例

调用方式：

外网域名映射HAproxy IP

### MySQL集群

采用一台主节点，多台从节点的方式部署，主节点主要负责录入dubbo业务组件采集的数据，从节点采用HAproxy(开源代理软件)搭建从节点集群slave cluster服务读取量较大的dubbo组件

目前的系统数据量不需要垂直和水平拆分，单表行记录拆分水平线500万记录，根据后期业务数据增长量进行垂直拆分多库，多集群部署

1个cluster = 1个master+2个slave (1个集群5个数据库)

搭建步骤：

1. 安装HAproxy
2. 安装docker复制MySQL镜像

调用方式：

写入调用master ip:3306

读取调用HAproxy IP:3306

## 运营监控化

为方便运维人员管理应用服务器，采用Prometheus**、**zabbix、Sky Walking框架统一管理监控系统日常活动状态

### 日志监控

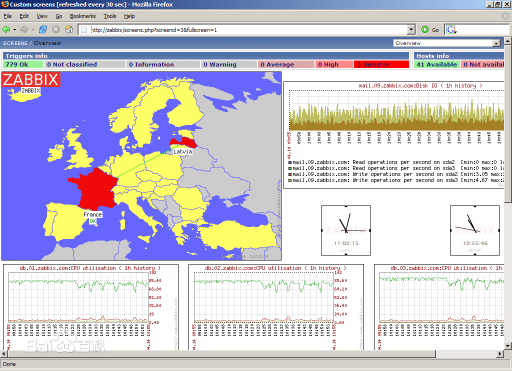
采用日志监控系统统一汇集和处理各个子系统的业务日志和系统日志

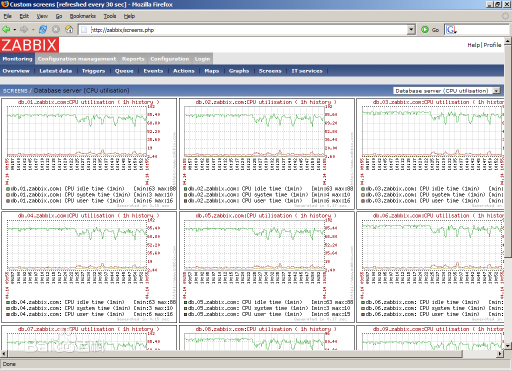
日志系统：

### 虚拟机硬件监控

1、采用分布式开源软件Zabbix，监控服务器的 CPU负荷、 内存使用、磁盘使用、网络状况、端口监视、 日志监视

2、服务器预警警告通过email方式通知运维人员





搭建步骤：

### Nginx运行监控

采用分布式开源软件Zabbix来监控Nginx的吞吐率、并发连接数、并发连接数详细统计，包括读取请求、处理请求和发送响应、持久连接、持久连接利用率等运行状态

搭建步骤：

1、Nginx启用ngx\_status配置项

在默认主机里面加上location或者你希望能访问到的主机里面

server {

    listen  \*:80 default\_server;

    server\_name \_;

    location /ngx\_status

    {

        stub\_status on;

        access\_log off;

        #allow 127.0.0.1;

        #deny all;

    }

}

1. 重启nginx ，service nginx restart
2. zabbix客户端配置，编写客户端脚本ngx\_status.sh

#!/bin/bash

# DateTime: 2015-10-25

# AUTHOR：凉白开

# WEBSITE: http://www.ttlsa.com

# Description：zabbix监控nginx性能以及进程状态

# Note：此脚本需要配置在被监控端，否则ping检测将会得到不符合预期的结果

# 文章地址：<span id="sample-permalink" tabindex="-1">http://www.ttlsa.com/zabbix/<span id="editable-post-name" title="临时固定链接，点击可更改这部分。">zabbix-monitor-nginx-performance</span>/</span> ‎

HOST="127.0.0.1"

PORT="80"

# 检测nginx进程是否存在

function ping {

    /sbin/pidof nginx | wc -l

}

# 检测nginx性能

function active {

    /usr/bin/curl "http://$HOST:$PORT/ngx\_status/" 2>/dev/null| grep 'Active' | awk '{print $NF}'

}

function reading {

    /usr/bin/curl "http://$HOST:$PORT/ngx\_status/" 2>/dev/null| grep 'Reading' | awk '{print $2}'

}

function writing {

    /usr/bin/curl "http://$HOST:$PORT/ngx\_status/" 2>/dev/null| grep 'Writing' | awk '{print $4}'

}

function waiting {

    /usr/bin/curl "http://$HOST:$PORT/ngx\_status/" 2>/dev/null| grep 'Waiting' | awk '{print $6}'

}

function accepts {

    /usr/bin/curl "http://$HOST:$PORT/ngx\_status/" 2>/dev/null| awk NR==3 | awk '{print $1}'

}

function handled {

    /usr/bin/curl "http://$HOST:$PORT/ngx\_status/" 2>/dev/null| awk NR==3 | awk '{print $2}'

}

function requests {

    /usr/bin/curl "http://$HOST:$PORT/ngx\_status/" 2>/dev/null| awk NR==3 | awk '{print $3}'

}

# 执行function

$1

1. zabbix客户端配置
2. 将自定义的UserParameter加入配置文件，然后重启agentd，如下：

#cat /usr/local/zabbix-3.0.0/etc/zabbix\_agentd.conf | grep nginx

UserParameter=nginx.status[\*],/usr/local/zabbix-3.0.0/scripts/ngx-status.sh $1

# killall zabbix\_agentd

# /usr/local/zabbix-3.0.0/sbin/zabbix\_agentd

#cat /usr/local/zabbix-3.0.0/etc/zabbix\_agentd.conf | grep nginx

UserParameter=nginx.status[\*],/usr/local/zabbix-3.0.0/scripts/ngx-status.sh $1

# killall zabbix\_agentd

# /usr/local/zabbix-3.0.0/sbin/zabbix\_agentd

zabbix\_get获取数据

此步骤可以跳过，但是最好是测试一下，因为通过此命令我们可以检测配置是否正确

# /usr/local/zabbix-3.0.0/bin/zabbix\_get -s 10.10.1.121 -k 'nginx.status[accepts]'

9570756

# /usr/local/zabbix-3.0.0/bin/zabbix\_get -s 10.10.1.121 -k 'nginx.status[ping]'

# /usr/local/zabbix-3.0.0/bin/zabbix\_get -s 10.10.1.121 -k 'nginx.status[accepts]'

9570756

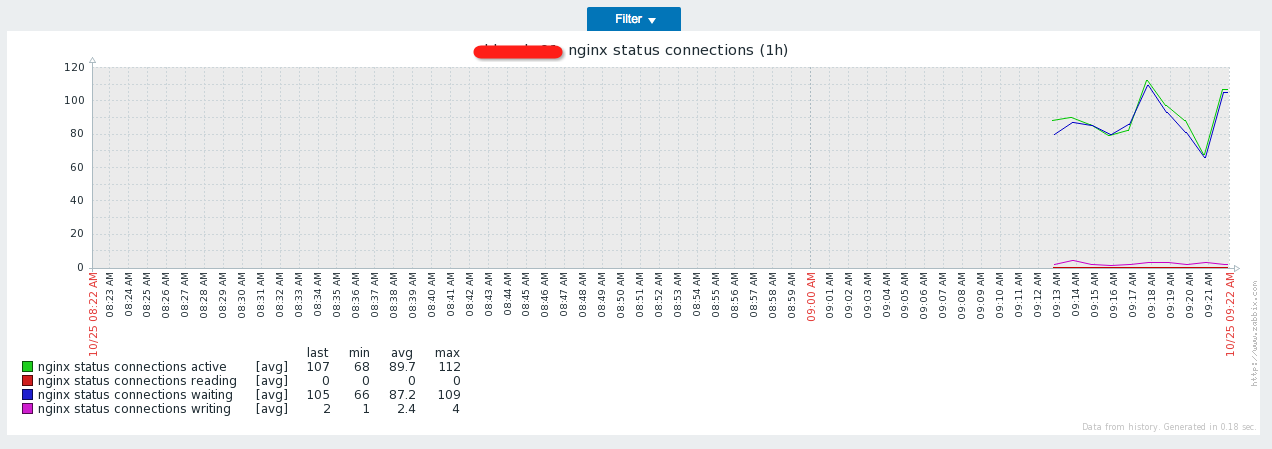
# /usr/local/zabbix-3.0.0/bin/zabbix\_get -s 10.10.1.121 -k 'nginx.status[ping]'

在zabbix server服务器上执行如上命令, 10.10.1.121为agentd机器

zabbix web端配置

导入Template App NGINX模板

调用方式：



### Tomcat运行监控

监控指标

JVM内存(包括JVM可使用内存、JVM所使用内存、JVM最大可使用内存)；

Tomcat请求数，包括每秒请求数，每秒出错数；

Tomcat网络流量统计，包括进流量统计，出流量统计；

Tomcat线程，包括最大线程数，当前线程数，当前繁忙线程数；

Tomcat处理时间，包括最大处理时间，平均处理时间；

采用Sky Walking框架监控Tomcat容器的跟踪查询、跨度细节、实例概述、JVM详细信息、服务依赖树、自动集成测试报告、性能测试报告

搭建步骤：

1. 安装Elasticsearch

修改 elasticsearch.yml

组 cluster.name: CollectorDBCluster

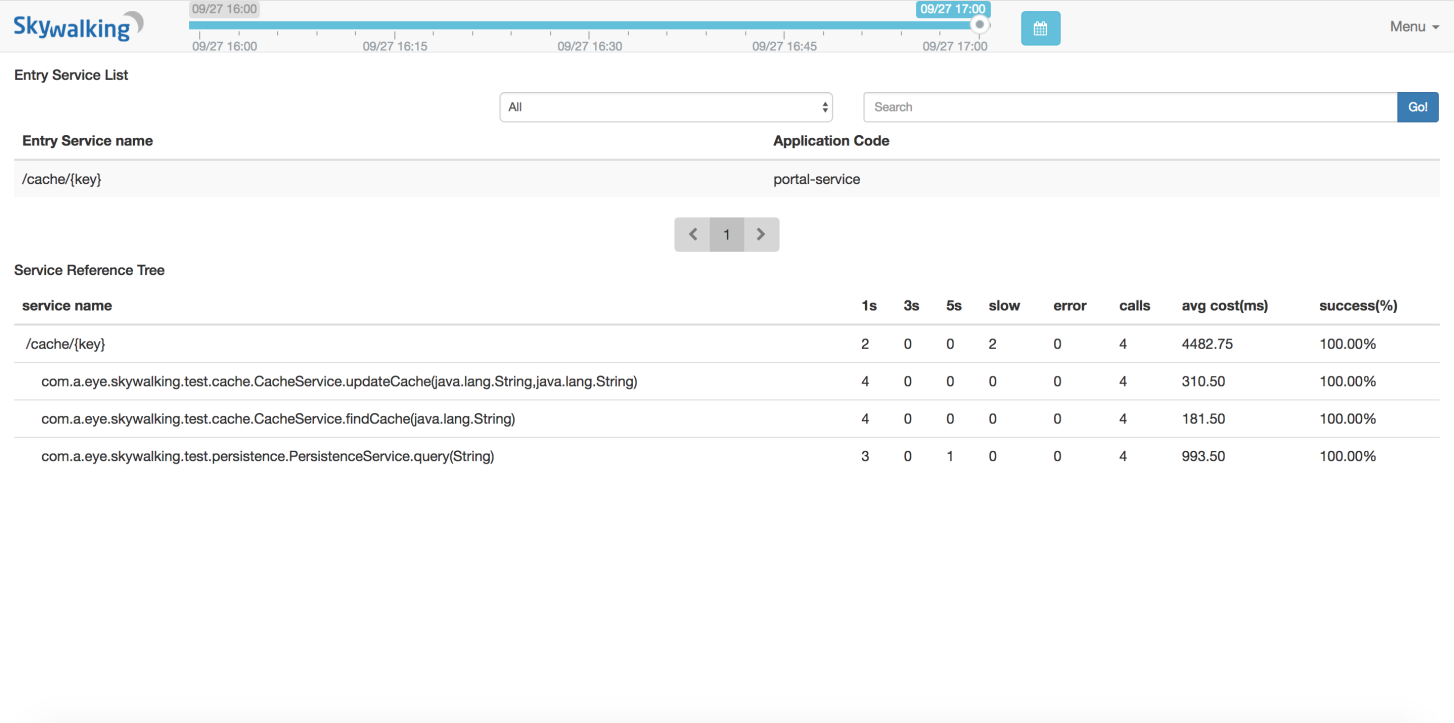
* 设置node.name: anyname，这个名字可以是任何的，它基于Elasticsearch。
* 将以下配置添加到

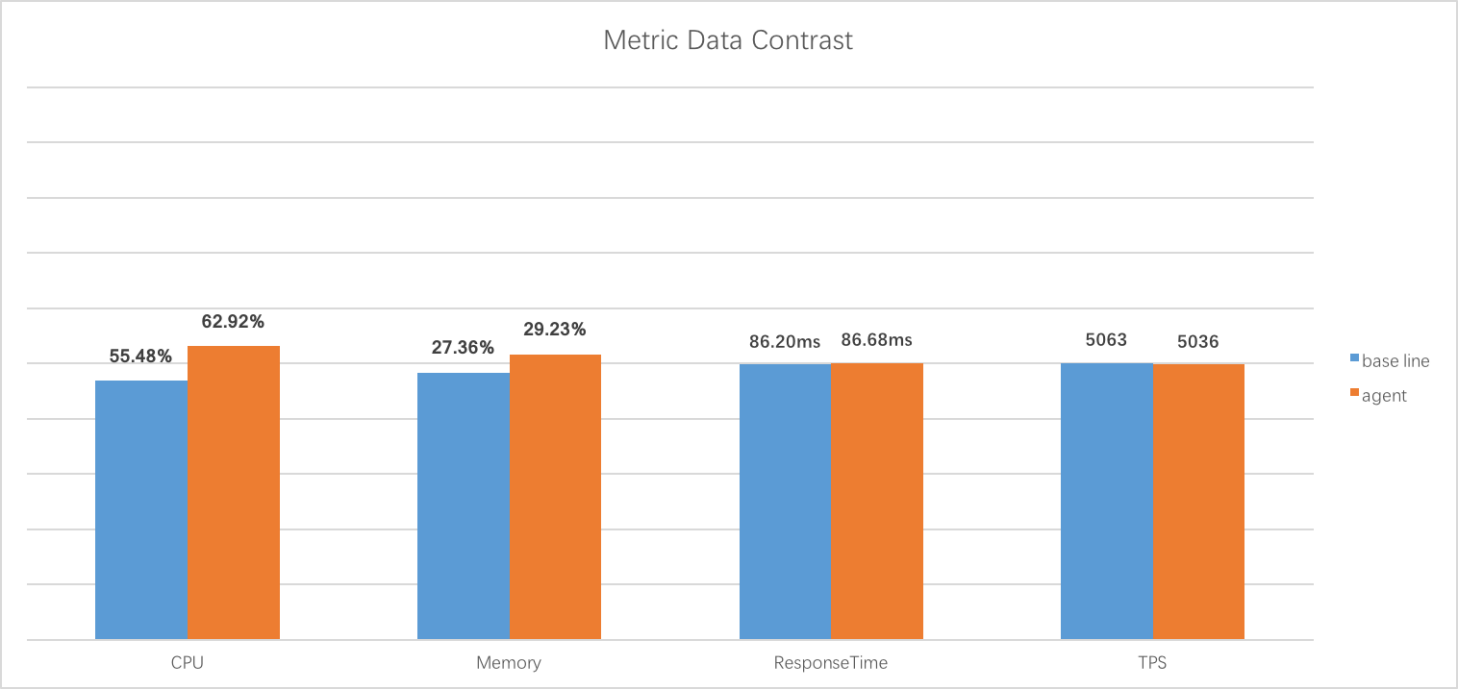
1. 安装Sky Walking

* tar -xvf skywalking-collector.tar.gz 在Linux中，或在Windows中解压缩。
* 运行bin/startup.sh或bin/startup.bat

调用方式







### Dubbo运行监控

监控指标

JVM内存(包括JVM可使用内存、JVM所使用内存、JVM最大可使用内存)；

Sky Walking框架监控Dubbo组件的跟踪查询、跨度细节、实例概述、JVM详细信息、服务依赖树、自动集成测试报告、性能测试报告

搭建步骤：

1. 安装Elasticsearch

修改 elasticsearch.yml

组 cluster.name: CollectorDBCluster

* 设置node.name: anyname，这个名字可以是任何的，它基于Elasticsearch。
* 将以下配置添加到

1. 安装Sky Walking

* tar -xvf skywalking-collector.tar.gz 在Linux中，或在Windows中解压缩。
* 运行bin/startup.sh或bin/startup.bat

调用方式：

通过配置收集的应用IP和端口，来搜集应用程序的调用数据

### Redis 运行监控

采用Zabbix官方模板监控Redis运行状况

搭建步骤：

下载地址：<https://github.com/blacked/zbx_redis_template>

1. **运行环境配置**

修改Host文件

vim /etc/hosts 添加 172.16.1.172    L-172.16.1.172

安装Python依赖包

[root@localhost /]# yum -y install python-pip

[root@localhost /]# pip install argparse

[root@localhost /]# pip install redis

1. 配置

1、 将zbx\_redis\_status.py上传至/etc/zabbix/script/redis目录并修改执行权限：

[root@localhost /]# mkdir -p /etc/zabbix/script/redis

[root@localhost /]# scp /usr/local/src/ zbx\_redis\_template-master/zbx\_redis\_stats.py 172.16.1.172:/etc/zabbix/script/redis

[root@localhost /]# chmod +x /etc/zabbix/script/redis/zbx\_redis\_stats.py

2、 根据实际情况修改zbx\_redis\_status.py中的下列参数：

zabbix\_host = '172.16.1.186' # Zabbix Server IP

zabbix\_port = 10051 # Zabbix Server Port

3、 测试zbx\_redis\_status.py是否可以正常连接数据库：

[root@localhost /]# /etc/zabbix/script/redis/zbx\_redis\_stats.py -h localhost -p 6379 -a MyPassword

前方有坑，请小心：

如果redis服务运行端口为其它值，请自行修改

如果出现如下图所示的提示，说明可以正常连接数据库，如果出现其他错误，请自行排错。

4、 将zbx\_redis.conf文件上传至/etc/zabbix/zabbix\_agentd.d目录:

[root@localhost /]# scp /usr/local/src/zbx\_redis\_template-master/zbx\_redis\_stats.py 172.16.1.172:/etc/zabbix/script/redis

5、 修改zbx\_redis.conf中的参数：

UserParameter=redis[\*],/etc/zabbix/script/redis/zbx\_redis\_stats.py –p 6379 -a MyPassword $1 $2 $3

前方有坑，请小心：

原配置文件是没有” -a MyPassword”的，如果登录redis需要提供密码，就要加上这个选项。如果登录redis不需要密码，则不需要添加。

原配置文件也没有” –p 6379”的，如果redis服务运行端口为其它值，请自行修改

6、 检查zabbix\_agentd.conf配置文件是否包含如下配置，如没有请自行添加：

Include=/etc/zabbix/zabbix\_agentd.d/

7、 测试脚本是否可以正常运行：

[root@yqtrack-userredis02 zabbix]# /etc/zabbix/script/redis/zbx\_redis\_stats.py -p 6379 -a MyPassword L-172.16.1.172 gcc\_version none

如果取得返回值，就说明脚本正常运行

参数解释：

$1: L-172.16.1.172

$2: gcc\_version

$3: none

参数“-p 6379 -a MyPassword”，

根据redis服务运行的端口及登录是否提供密码，请根据实际情况确定

8、 在zabbix配置界面导入模板配置文件(zbx\_redis\_templates.xml)

1. 参考文档：README.md

调用方式：

### MQ运行监控

采用自定义脚本配合Zabbix来监控MQ 发布的消息总数、显示准备就绪的消息、未确认的消息、未提交的消息、显示还未必数据库处理的事件数量、消费者个数、队列长度、队列交换机的数量、连接数、频道数量等运行情况

搭建步骤：

<http://blog.51cto.com/john88wang/1745824>

调用方式

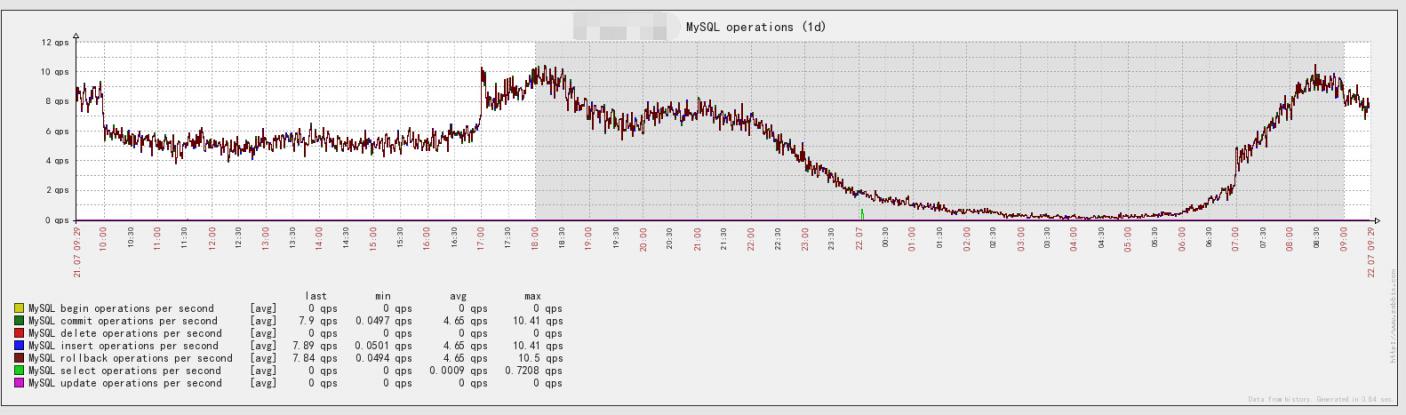
### MySQL运行监控

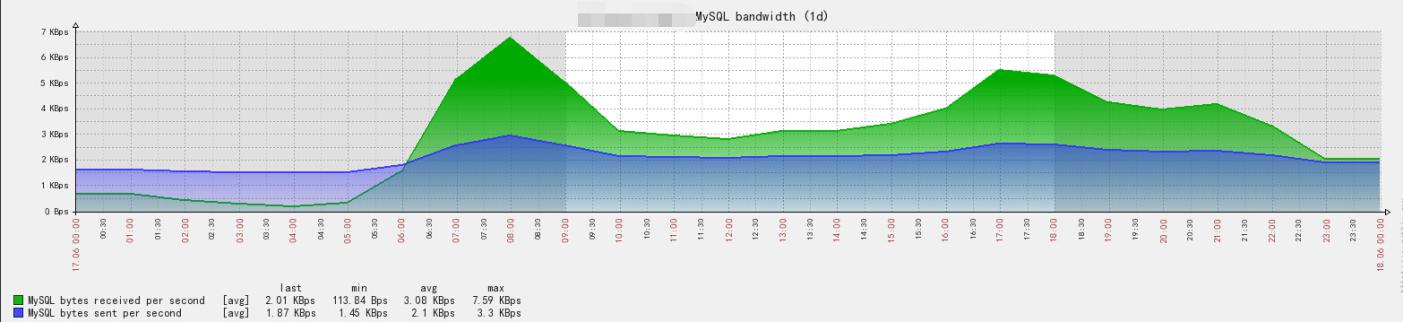
采用自定义脚本配合Zabbix来监控MySQL的请求流量带宽、响应流量带宽、ops、MySQL查询吞吐率(包括Change DB、Select、Insert、Update、Delete)、MySQL持久连接利用率、 MySQL查询缓存空间使用率、MySQL查询缓存命中率、MySQL缓存查询数、MySQL索引缓存命中率、MySQL索引读取统计、MySQL连接吞吐率、MySQL连接缓存命中率、MySQL并发连接数(包括最大允许连接数、实际最大连接数、当前连接数、活跃连接数、缓存连接数)、 MySQL流量统计、MySQL表统计锁定等运行状况

搭建步骤：

<http://www.ttlsa.com/zabbix/zabbix-monitor-mysql/>

调用方式：





### FAST DFS文件服务运行监控

搭建步骤：

## 升级部署平滑化

所有项目支持版本化，灰度发布，统一管理，统一升级

### 基础环境升级

centos升级

利用docker快速复制升级后的镜像

jdk升级

下载地址

利用docker快速复制升级后的镜像

mesos升级

下载地址

利用docker快速复制升级后的镜像

marthon升级

下载地址

利用docker快速复制升级后的镜像

Docker升级

下载地址

利用docker快速复制升级后的镜像

Zookeeper升级

下载地址

利用docker快速复制升级后的镜像

### 基础应用升级

HAproxy 升级

Nginx升级

下载新版Nginx源码

Tomcat容器升级

下载地址

利用docker快速复制升级后的镜像

Dubbo组件升级

下载地址

利用docker快速复制升级后的镜像

Zabbix升级

下载地址

利用docker快速复制升级后的镜像

### 业务应用升级

Web应用升级

业务组件升级

### 数据库升级

以集群方式升级

主节点统一升级，从节点复制升级

Redis升级

MySQL升级

memcached升级

RabbitMQ升级

### 项目的升级

统一节点构建，应用节点下载部署

### 紧急bug升级

### 快速升级

## 配置集中管理化

为简化开发、部署和维护系统配置参数的复杂度，更为隐藏数据库关键地址信息，采用当当的config-toolkit框架统一管理系统资源配置信息

### 单台虚拟节点系统配置

系统网络参数配置

系统端口配置

系统防火墙配置

系统内存配置

系统硬盘空间配置

### 基础环境配置

Jvm参数配置

$JAVA\_ARGS

-Xmx4000M  
-Xms4000M  
-Xmn600M  
-XX:PermSize=500M  
-XX:MaxPermSize=500M  
-Xss256K  
-XX:+DisableExplicitGC  
-XX:SurvivorRatio=1  
-XX:+UseConcMarkSweepGC  
-XX:+UseParNewGC  
-XX:+CMSParallelRemarkEnabled  
-XX:+UseCMSCompactAtFullCollection  
-XX:CMSFullGCsBeforeCompaction=0  
-XX:+CMSClassUnloadingEnabled  
-XX:LargePageSizeInBytes=128M  
-XX:+UseFastAccessorMethods  
-XX:+UseCMSInitiatingOccupancyOnly  
-XX:CMSInitiatingOccupancyFraction=80  
-XX:SoftRefLRUPolicyMSPerMB=0  
-XX:+PrintClassHistogram  
-XX:+PrintGCDetails  
-XX:+PrintGCTimeStamps  
-XX:+PrintHeapAtGC  
-Xloggc:log/gc.log

Jdbc参数配置

redis客户端配置

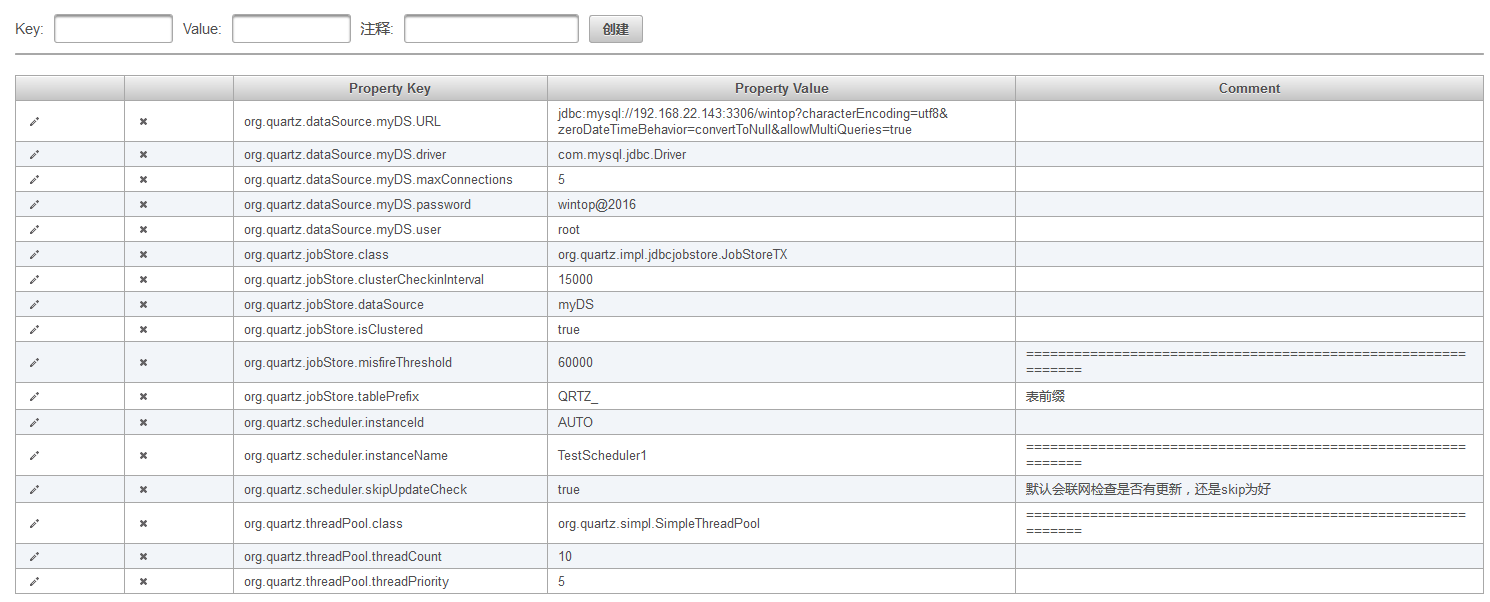
Memcached客户端配置

RabbitMQ客户端配置

mybatis客户端配置

Log4j配置

Quartz定时器参数配置



### 应用容器配置

Tomcat服务端基础配置

<Connector port="8080"  
protocol="org.apache.coyote.http11.Http11NioProtocol"  
connectionTimeout="20000"  
redirectPort="8443"  
maxThreads="5000"  
minSpareThreads="20"  
acceptCount="1000"  
disableUploadTimeout="true"  
enableLookups="false"  
URIEncoding="UTF-8" />  
Catalina.sh JVM 参数优化，添加如下内容：

CATALINA\_OPTS="$CATALINA\_OPTS –Xms4000M –Xmx4000M  
–Xmn1000M -XX:SurvivorRatio=4 -XX:+UseConcMarkSweepGC  
-XX:CMSInitiatingOccupancyFraction=82 -DLOCALE=UTF-16LE  
-DRAMDISK=/ -DUSE\_RAM\_DISK=ture -DRAM\_DISK=true"

Dubbo应用服务端基础配置

# 1.failover cluster 失败自动切换，当出现失败，重试其他服务器（缺省），通常用于读操作，但重试会带来更长的延时，可通过retries=“2”来设置重试次数（不含第一次） <dubbo:service retries="2"> 或者 <dubbo:reference retries="2"> 或者 <dubbo:reference> <dubbo:method name="findFoo" retries=2> <dubbo:reference/> 2.failfast cluster 快速失效，只发起一次调用，失败立即报错。通常用于非幂等性写操作，比如说新增记录 <dubbo:service cluster="failfast"> 或者 <dubbo:reference cluster="failfast" 3.failsaft cluster 失败安全，出现异常时，直接忽略，通常用于写入审计日志等操作 <dubbo:service cluster="failsafe"> 或者 <dubbo:reference cluster="failsafe"> 4.failback cluster 失败自动恢复，后台记录失败请求，定时重发，通常用于消息通知操作 <dubbo:service cluster="failback"> 或者 <dubbo:reference cluster="failback"> 5.forking cluster 并行调用多个服务器，只要一个成功即返回。通常用于实时性要求较高的读操作，但需要浪费更多的服务器资源。可通过forks=“2”来设置最大并行数。 <dubbo:service cluster="forking"> 或者 <dubbo:reference cluster="forking">

dubbo.provider.cluster=failfast

dubbo.provider.delay=-1

# 1、Random LoadBalance 随机，按权重设置随机概率。 在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。 2、RoundRobin LoadBalance 轮循，按公约后的权重设置轮循比率。 存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。 解决办法 :结合权重，把第二台机(性能低的)的权重设置低一点 3、LeastActive LoadBalance 最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。 使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。 4、ConsistentHash LoadBalance 一致性Hash，相同参数的请求总是发到同一提供者。 当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。 算法参见：http://en.wikipedia.org/wiki/Consistent\_hashing。 缺省只对第一个参数Hash，如果要修改，请配置<dubbo:parameter key="hash.arguments" value="0,1" /> 缺省用160份虚拟节点，如果要修改，请配置<dubbo:parameter key="hash.nodes" value="320" />

dubbo.provider.loadbalance=roundrobin

# Dispatcher all 所有消息都派发到线程池，包括请求，响应，连接事件，断开事件，心跳等。 direct 所有消息都不派发到线程池，全部在IO线程上直接执行。 message 只有请求响应消息派发到线程池，其它连接断开事件，心跳等消息，直接在IO线程上执行。 execution 只请求消息派发到线程池，不含响应，响应和其它连接断开事件，心跳等消息，直接在IO线程上执行。 connection 在IO线程上，将连接断开事件放入队列，有序逐个执行，其它消息派发到线程池。 ThreadPool fixed 固定大小线程池，启动时建立线程，不关闭，一直持有。(缺省) cached 缓存线程池，空闲一分钟自动删除，需要时重建。 limited 可伸缩线程池，但池中的线程数只会增长不会收缩。(为避免收缩时突然来了大流量引起的性能问题)。

dubbo.provider.threadpool=cached

# 服务注册中心地址

dubbo.registry.address=192.168.22.143:2181

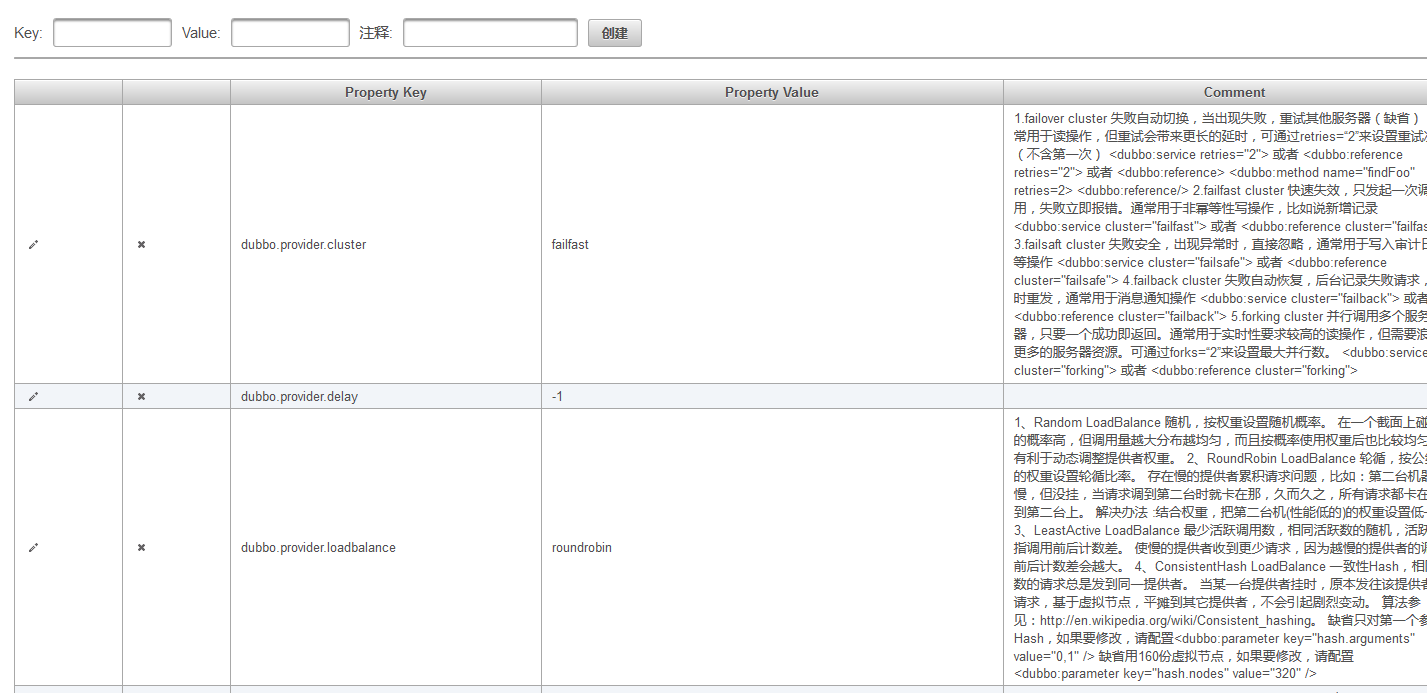
# 服务注册中心协议

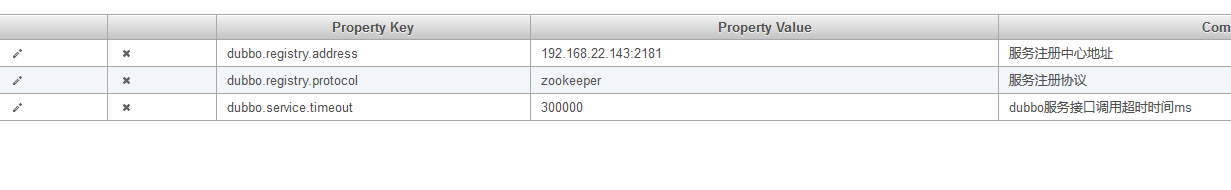
dubbo.registry.protocol=zookeeper

dubbo.service.cluster=failfast

dubbo.service.delay=-1

dubbo应用客户端基础配置





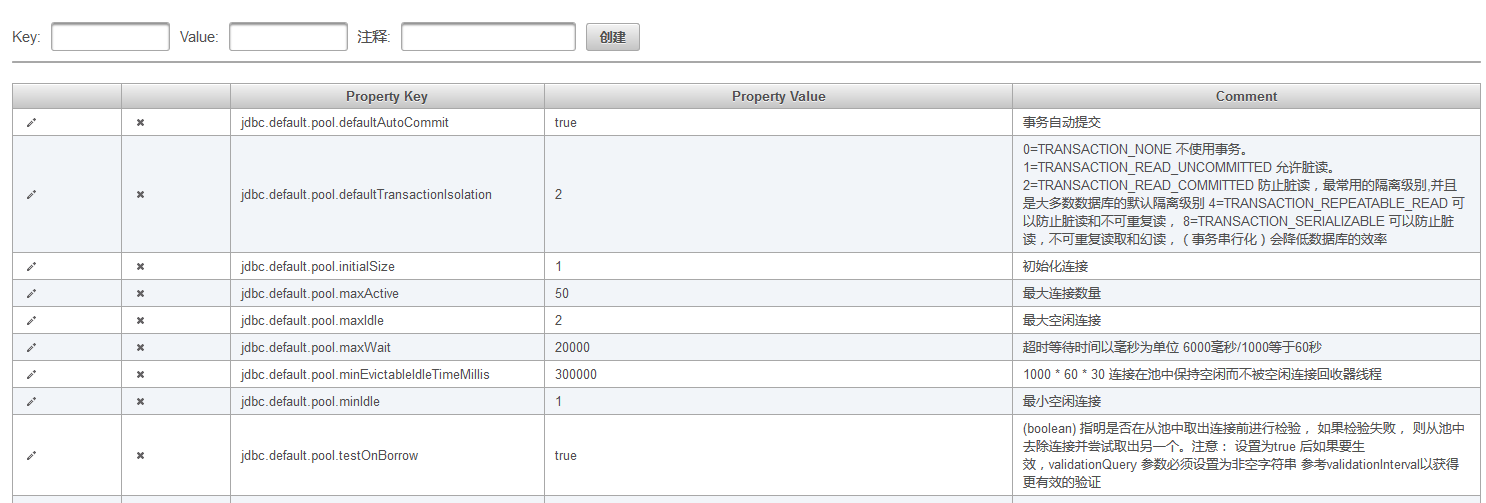
### 业务组件服务配置

Tomcat服务应用配置

Dubbo服务端应用配置

dubbo客户端应用配置

引用数据源配置



## 环境镜像化

采用docker开源应用容器镜像化系统平台基础应用，方便快速复制环境和快速搭建应用

### 虚拟机镜像

Docker-

### 单台虚拟节点系统docker镜像

方案：

步骤：

### 基础环境docker镜像

方案：jdk集成到docker里

步骤：

### 业务应用docker镜像

方案

步骤：

## 项目版本化

每次升级需要定义版本号、升级明细

命名规则

1.0.0主版本号.子版本号[.编译版本号[.修正版本号]]

\* Base版: 此版本表示该软件仅仅是一个假页面链接，通常包括所有的功能和页面布局，但是页面中的功能都没有做完整的实现，只是做为整体网站的一个基础架构。  
\* Alpha版: 此版本表示该软件在此阶段主要是以实现软件功能为主，通常只在软件开发者内部交流，一般而言，该版本软件的Bug较多，需要继续修改。  
\* Beta版: 该版本相对于Alpha版已有了很大的改进，消除了严重的错误，但还是存在着一些缺陷，需要经过多次测试来进一步消除，此版本主要的修改对像是软件的UI。  
\* RC版: 该版本已经相当成熟了，基本上不存在导致错误的BUG，与即将发行的正式版相差无几。  
\* Release版: 该版本意味“最终版本”，在前面版本的一系列测试版之后，终归会有一个正式版本，是最终交付用户使用的一个版本。该版本有时也称为标准版。一般情况下，Release不会以单词形式出现在软件封面上，取而代之的是符号(Ｒ)。