**贵交科Java开发手册**

**（初版）**

贵州智行通科技有限公司

2018-03-15

目录

[1 开发规范 2](#_Toc509042615)

[1.1 编程规范 2](#_Toc509042616)

[1.2 数据交互约定 2](#_Toc509042617)

[2 Maven 5](#_Toc509042618)

[2.1 Maven下载安装 5](#_Toc509042619)

[2.2 Maven配置 5](#_Toc509042620)

[2.3 Maven使用 5](#_Toc509042621)

[3 Git客户端使用 5](#_Toc509042622)

[3.1 Gitlab客户端安装 5](#_Toc509042623)

[3.2 Git可视化工具之tortoise 6](#_Toc509042624)

[3.3 tortoisegit汉化说明 6](#_Toc509042625)

[3.4 tortoisegit常用操作 6](#_Toc509042626)

[4 Spring boot 6](#_Toc509042627)

[5 Spring cloud 6](#_Toc509042628)

[5.1 Config（配置中心） 6](#_Toc509042629)

[5.1.1 编写configserver 7](#_Toc509042630)

[5.1.2 编写configclient 7](#_Toc509042631)

[5.2 Eureka （注册中心） 8](#_Toc509042632)

[5.2.1 编写eureka server 8](#_Toc509042633)

[5.2.2 eureka server加上访问权限 9](#_Toc509042634)

[5.2.3 编写eureka client 9](#_Toc509042635)

[5.3 Zuul （网关） 9](#_Toc509042636)

[5.3.1 编写zuul网关 10](#_Toc509042637)

[5.3.2 Zuul路由规则 10](#_Toc509042638)

[5.3.3 Zuul过滤器 11](#_Toc509042639)

[5.4 熔断保护 11](#_Toc509042640)

[5.4.1 熔断监控 11](#_Toc509042641)

[5.5 负载均衡 13](#_Toc509042642)

[5.6 服务之间的通信 13](#_Toc509042643)

[5.7 消息总线 14](#_Toc509042644)

[5.8 使用Spring Cloud Sleuth实现微服务跟踪 15](#_Toc509042645)

[5.8.1 整合Spring Cloud Sleuth 15](#_Toc509042646)

[5.8.2 Spring Cloud Sleuth与ELK配合使用 15](#_Toc509042647)

[5.8.3 Spring Cloud Sleuth与Zipkin配合使用 15](#_Toc509042648)

[6 Docker部署运用 15](#_Toc509042649)

[6.1 Docker 安装 16](#_Toc509042650)

[6.2 Docker 构建镜像 17](#_Toc509042651)

[6.3 dockerfile编写 17](#_Toc509042652)

[6.4 DockerCompose安装 17](#_Toc509042653)

[6.5 DockerCompose编排微服务 17](#_Toc509042654)

[6.6 配置Jenkins自动更新代码 18](#_Toc509042655)

[7 贵交科服务架构设计说明 18](#_Toc509042656)

# 开发规范

## 编程规范

1. 代码中允许中英文混编，不能缩写，不能用中文，尽量让别人容易理解。
2. 代码中必须加注释
3. 必须以字母开头并且遵守驼峰式命名规则
4. 类名首字母大写。方法名、参数名、成员变量、局部变量首字母统一小写。常量所有字母大写，单词之间用下划线连接。包名使用全小写，单词之间用点分隔
5. 抽象类命名以base开头。异常以Exception结尾。测试类以类名开头Test结尾
6. 布尔类型的命名前面不要加is
7. 接口类型的方法和属性不要加任何修饰符包括public都不要加，但是必须加上注释。不允许在接口类里面定义变量
8. 枚举类型以Enum结尾，成员名称全部大写，用下划线隔开，必须加上注释
9. 统一使用三层结构，api层所有类必须以controller结尾。逻辑处理层 接口类必须以service结尾，实现类必须以接口类名开头 Impl结尾。数据持久化 接口类Dao结尾 实现类以接口类名开头Impl结尾
10. 方法命名增加必须以save开头，删除必须以remove开头，修改必须以update开头，查询必须以query开头
11. 不允许在使用中出现写死的常量，常量必须放置在常量类里面，不要使用一个常量类管理所有常量，应该使用对应功能对应不同的常量类。常量类命名应以constant结尾。
12. 对外api层必须放在controller包路径下，数据持久层必须放在dao包下边，业务逻辑层必须放在service包路径下边，常量必须放到constant包下边，工具类必须放到util下。对应功能自己建子包管理。

## 数据交互约定

1. 和前端交互所有返回的json格式必须为

{code:200,msg:”说明文字”,data:”具体数据”}

1. 所有异常不允许抛给前端，给前端的永远只允许用返回码
2. 状态码code说明

|  |  |  |
| --- | --- | --- |
| 状态码 | 说明 | 备注 |
| 200-299 | 成功相关信息 |  |
| 300-399 |  |  |
| 400-499 | 服务器逻辑错误 |  |
| 500-599 | 服务器内部错误 |  |

具体使用说明

|  |  |  |
| --- | --- | --- |
| 200 | 成功处理并返回相关信息 |  |
| 400 | 错误请求 |  |
| 401 | 未授权 |  |
| 402 | 过期的api 并返回新的API文档地址 |  |
| 403 | 验证失败 |  |
| 404 | 不存在的API |  |
| 500 | 服务器错误 |  |

# Maven

## Maven下载安装

下载地址：<http://maven.apache.org/download.cgi>

安装自行解决

## Maven配置

找到安装目录conf找到配置文件setting.xml打开编辑，设置本地仓库和中央仓库。

本地仓库设置：localRepository>E:\java\mavenrep</localRepository>

中央仓库设置：

<mirror>

<id>alimaven</id>

<name>aliyun maven</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>

<mirrorOf>central</mirrorOf>

</mirror>

## Maven使用

打开编译器以STS为例: 找到window---->preferences---->maven---->installations

添加自己的maven安装信息

找到window---->preferences---->maven---->User Setting

设置自己的maven安装路径即可

# Git客户端使用

## Gitlab客户端安装

下载地址为：<https://git-scm.com/downloads/>

安装自行解决

## Git可视化工具之tortoise

百度一下tortoisegit下载安装自行解决

## tortoisegit汉化说明

百度tortoisegit中文语言包，安装自行解决。打开设置，找到general🡪language选择简体中文就好

## tortoisegit常用操作

获取git项目，在你想要项目存放的位置，右键git clone 把仓库的项目地址复制进去就点击确认等待下载。提交项目前请先拉取在提交，提交必须填写提交说明才能提交

# Spring boot

详情请见springboot中文手册

# Spring cloud

springCloud是基于SpringBoot的一整套实现微服务的框架。他提供了微服务开发所需的配置管理、服务发现、断路器、智能路由、微代理、控制总线、全局锁、决策竞选、分布式会话和集群状态管理等组件。最重要的是，跟spring boot框架一起使用的话，会让你开发微服务架构的云服务非常的方便。

## Config（配置中心）

在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中（根据实际情况，我们选择的是放到自己私有的gitlab仓库中）。在spring cloud config 组件中，分两个角色，一是config server，二是config client。工作原理如图5.1所示

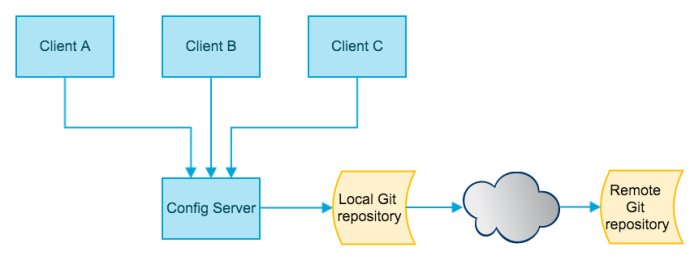


图5.1 配置中心工作原理图解

### 编写configserver

Maven 引入jar包

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-config-server</artifactId>**

**</dependency>**

**添加注解：@EnableConfigServer**

**添加配置：**

**spring:**

**application:**

**name: config**

**cloud:**

**config:**

**server:**

**git:**

**uri: 配置仓库的git地址**

**username: 授权用户名**

**password: 密码**

### 编写configclient

Maven 引入jar包

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-config</artifactId>**

**</dependency>**

**添加注解：@EnableEurekaClient**

**添加配置：**

**spring:**

**cloud:**

**config:**

**uri: 访问config的地址和端口号**

**fail-fast:** true

**application:**

**name: 工程名**

注：config client的配置文件必须为bootstrap.yml否则无法加载配置

## Eureka （注册中心）

Eureka是Netflix开发的服务发现框架，SpringCloud将它集成在自己的子项目spring-cloud-netflix中，实现SpringCloud的服务发现功能。

为什么要使用Eureka，因为在一个完整的系统架构中，任何单点的服务都不能保证不会中断，因此我们需要服务发现机制，在某个节点中断后，其它的节点能够继续提供服务，从而保证整个系统是高可用的。 服务发现有两种模式：一种是客户端发现模式，一种是服务端发现模式。Erueka采用的是客户端发现模式。工作流程如图5.2所示

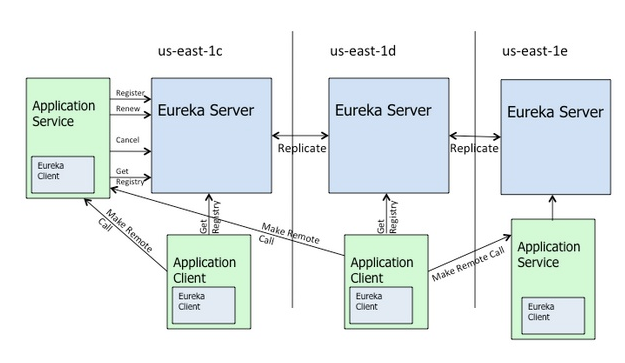


图5.2 eureka的工作流程

### 编写eureka server

**Maven 引入jar包：**

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-eureka-server</artifactId>**

**</dependency>**

**启动类上边加上注解@EnableEurekaServer**

**添加配置文件**

eureka:

server:

enableSelfPreservation: false #关闭自我保护

evictionIntervalTimerInMs: 4000 #清理服务时间（默认60\*1000）

### eureka server加上访问权限

maven 引入jar包

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

添加配置文件：

security:

basic:

enabled: true

user:

name: 授权用户名

password: 授权密码

### 编写eureka client

Maven添加依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

Main类上边添加注解：@EnableEurekaClient

添加配置文件：

eureka:

instance:

prefer-ip-address: true

instance-id: ${spring.application.name}:${spring.cloud.client.ipAddress}:${spring.application.instance\_id:${server.port}}

lease-renewal-interval-in-seconds: 5

client:

service-url:

defaultZone: http://用户名:密码@registry:18761/eureka

注：配置文件应该添加在配置中心的文件中。查看是否注册成功，访问eureka的地址和启动端口号加上eureka的路径即可如：<http://localhost:8761/eureka>

## Zuul （网关）

不同的微服务一般会有不同的网络地址，而外部客户端可能需要调用多个服务的接口才能完成一个业务需求。如：一个电影购票的手机APP，可能会调用多个微服务，才能完成一次购票的业务流程。如果让客户端直接与各个微服务通信，会有以下的问题：

1. 客户端会多次请求不同的微服务，增加了客户端的复杂性；
2. 存在跨域请求，在一定场景下处理相对复杂；
3. 认证复杂，每个服务都需要独立认证；
4. 难以重构，随着项目的迭代，可能需要重新划分微服务，如果客户端直接与微服务通信，那么重构将会很难实施；
5. 某些微服务可能使用了防火墙/浏览器不友好的协议，直接访问会有一定的困难。

以上问题可借助微服务网关解决，微服务网关是介于客户端和服务端之间的中间层，所有的外部请求都会先经过微服务网关，然后由微服务网关请求各个微服务。

微服务网关封装了应用程序的内部结构，客户端只须跟网关交互，而无须直接调用特定微服务的接口，这样，开发就可以得到简化。不仅如此，使用微服务网关还有以下优点：

1. 易于监控。可在微服务网关收集监控数据并将其推送到外部系统进行分析；
2. 易于认证。可在微服务网关上进行认证，然后再将请求转发到后端的微服务，而无须再每个微服务中进行认证；
3. 减少了客户端与各个微服务之间的交互次数

zuul工作图解如图5.3所示

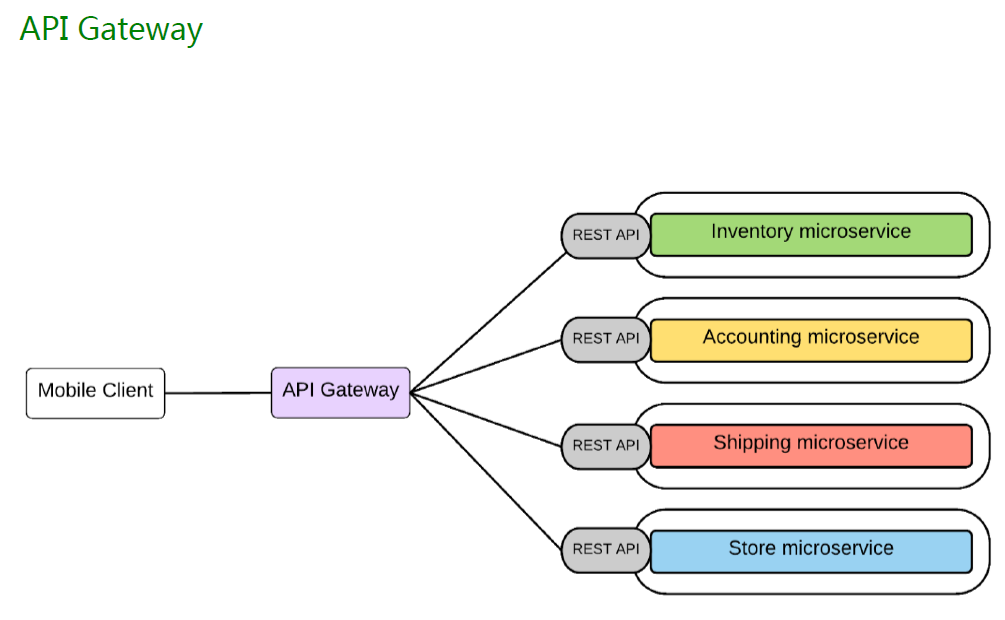


图5.3 zuul工作图

### 编写zuul网关

Maven 引入jar包

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-zuul</artifactId>**

**</dependency>**

**添加注解：@EnableZuulProxy**

### Zuul路由规则

添加路由规则配置即可如下：

#zuul:

# ignoredServices: '\*'

# host:

# connect-timeout-millis: 20000

# socket-timeout-millis: 20000

# routes:

# driver-server:

# path: /driver/\*\*

# serviceId: driver

# stripPrefix: false

# sensitiveHeaders:

# passenger-server:

# path: /passenger/\*\*

# serviceId: passenger

# stripPrefix: false

# sensitiveHeaders:

### Zuul过滤器

因为这个要说清楚就必须贴代码，然而我并不想贴代码在文档里面，因为太站篇幅，其二这篇博客说的挺详细的，所以借用这篇博客加以说明。<http://blog.csdn.net/liuchuanhong1/article/details/62236793>

## 熔断保护

为保证服务高可用，最先想到的是服务集群，但集群并不能完全的保证服务高可用，当某个服务出现故障时，在负载均衡的时候可能多次被调用到，调用方由于无法得到调用结果，会出现请求超时会其他异常，这时候如果不及时的熔断服务，就有可能会有更多的调用者去调用已经出现故障的服务节点，造成大量调用失败，甚至引发联级故障的雪崩。

熔断，就是断开与服务器的连接，熔断器是在服务不可用的时候主动断开，以免造成更多的雪崩效应，他是保护服务高可用的最后一道防线。如图5.4所示

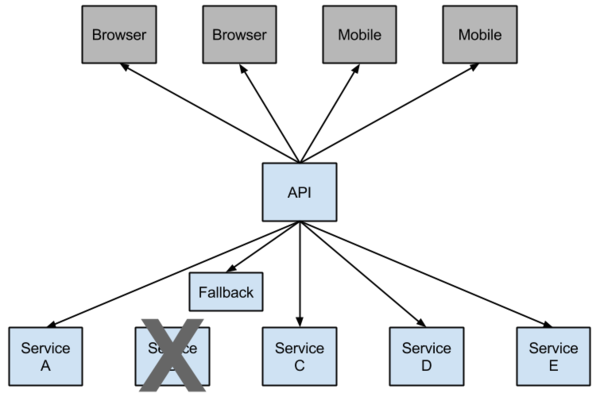


图5.4熔断保护

### 熔断监控

1. Dashboard

Maven引入jar包：

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

加入注解：

@EnableHystrix

@EnableHystrixDashboard

@EnableCircuitBreaker

在接口上面加上熔断处理相关操作

@HystrixCommand(fallbackMethod = "hiError")

访问[http://localhost:8080/hystrix 即可查看监控如图5.4.1](http://localhost:8080/hystrix%20即可查看监控如图5.4.1)所示

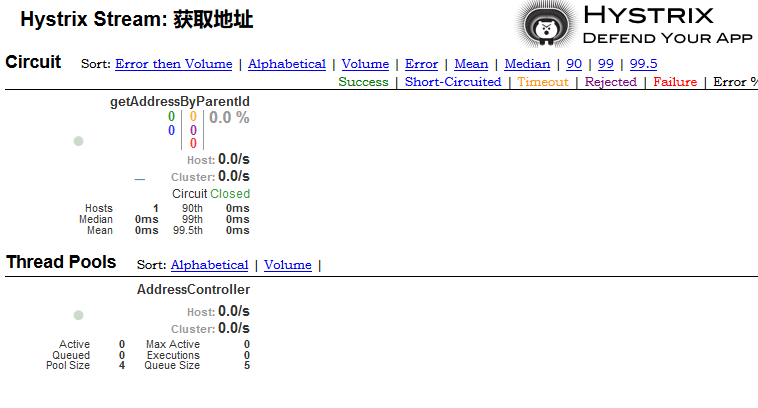


图5.4.1 Dashboard服务监控

1. Turbine

Maven引入jar包：

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-turbine</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-netflix-turbine</artifactId>

</dependency>

添加注解：@EnableTurbine

添加配置：其实就是配置多个节点聚合成为一个面板

## 负载均衡

Ribbon做负载均衡，主要是如下两个步骤

1. 第一步有限选择Eureka Server，它优先选择在同一个Zone且负载较少的Server，
2. 第二步在根据用户指定的策略，在从Server取到的服务注册列表中选择一个地址。其中Ribbon提供了多重策略，例如轮询round robin、随机Random、根据相应时间加权等。

具体工作如图5.5所示

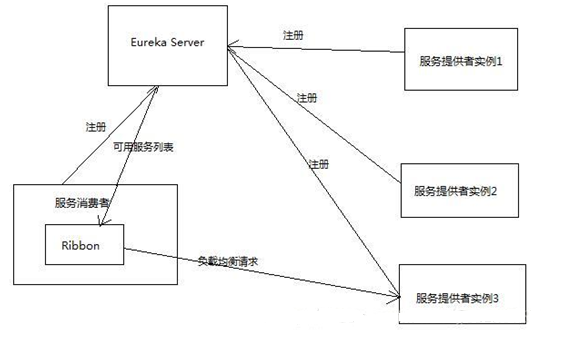


图5.5 Robbin负载均衡机制

## 服务之间的通信

微服务之间的通讯建议是用fegin，下面简单介绍一下fegin的使用

Maven引入fegin的jar包：

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-feign</artifactId>

</dependency>

启动类加上注解：@EnableFeignClients

编写fegin的接口类并且加上注解：@FeignClient(name ="目标服务的服务名称")

定义需要访问的接口信息如：

@RequestMapping(value ="/addOrderNum", method = RequestMethod.***POST***)

**public** **void** addOrderNum(@RequestParam("driverPhone")String driverPhone);

fegin默认已经集成了ribbon做负载均衡，默认负载机制为轮询负载

## 消息总线

首先需要安装rabbitmq，然后maven引入jar包：

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

配置mq相关信息：

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

spring.rabbitmq.username=springcloud

spring.rabbitmq.password=123456

工作原理如图5.7所示

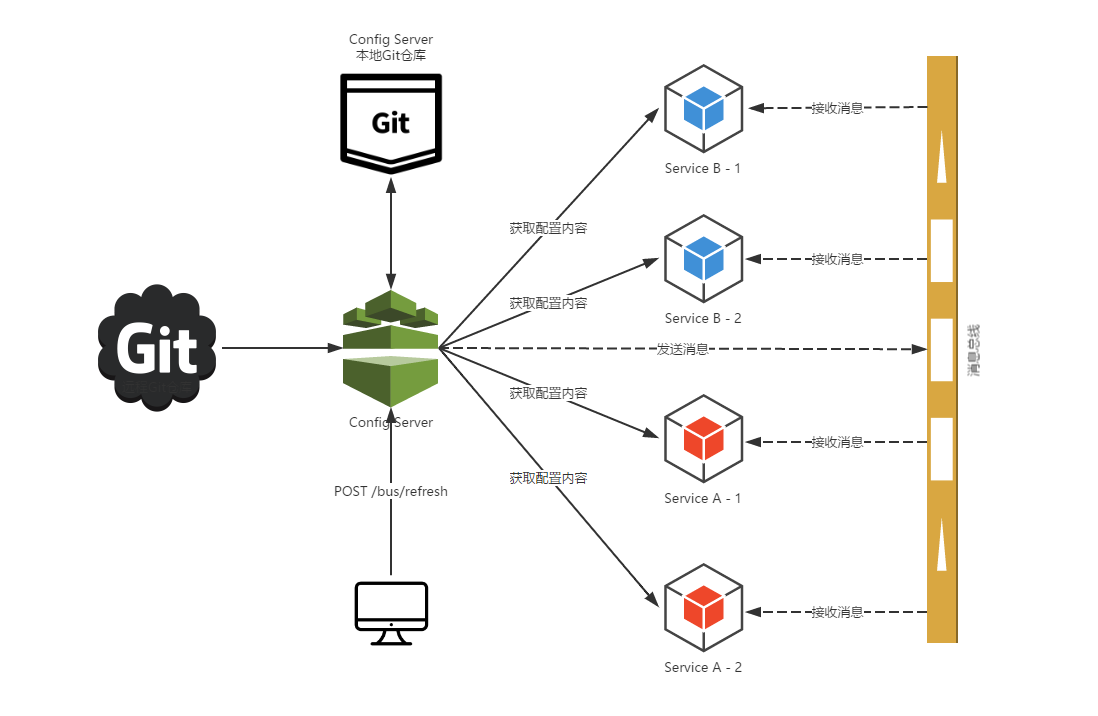


图5.7消息总线工作原理

## 使用Spring Cloud Sleuth实现微服务跟踪

### 整合Spring Cloud Sleuth

### Spring Cloud Sleuth与ELK配合使用

### Spring Cloud Sleuth与Zipkin配合使用

1. 编写Zipkin Server
2. 微服务整合Zipkin
3. 使用消息中间件收集数据
4. 存储跟踪数据（Elasticsearch 5.1.2）

# Docker部署运用

首先，Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多。 其次，Docker 对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器。

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。

具体说来，Docker 在如下几个方面具有较大的优势。

1. 更快速的交付和部署

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。

1. 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。

1. 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。

1. 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

Docker 工作流程如图6所示

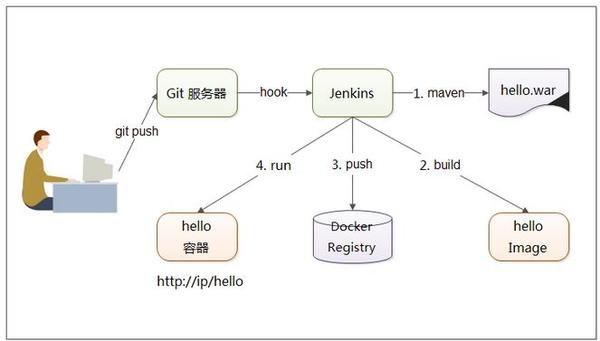


图6 docker使用流程

## Docker 安装

1. 查看系统版本：

$ cat /etc/redhat-release

CentOS Linux release 7.0.1406 (Core)

1. 安装docker

$ yum install docker

安装过程中报错：

Transaction check error:

file /usr/lib/systemd/system/blk-availability.service from install of device-mapper-7:1.02.107-5.el7\_2.2.x86\_64 conflicts with file from package lvm2-7:2.02.105-14.el7.x86\_64

file /usr/sbin/blkdeactivate from install of device-mapper-7:1.02.107-5.el7\_2.2.x86\_64 conflicts with file from package lvm2-7:2.02.105-14.el7.x86\_64

file /usr/share/man/man8/blkdeactivate.8.gz from install of device-mapper-7:1.02.107-5.el7\_2.2.x86\_64 conflicts with file from package lvm2-7:2.02.105-14.el7.x86\_64

解决办法：$yum install libdevmapper\* -y

再次运行：$yum install docker

1. 检查安装是否成功

$docker version

若输出了 Docker 的版本号，说明安装成功了，可通过以下命令启动 Docker 服务：

1. $service docker start

一旦 Docker 服务启动完毕，就**可以开始使用 Docker 了。**

## Docker 构建镜像

1. 查看运行容器列表docker ps
2. 查看全部容器列表 docker ps -a
3. 查看镜像docker images
4. 删除镜像docker rmi 镜像ID
5. 构建镜像docker build -t="镜像名称" . 注意：后面的那个点不能丢

## dockerfile编写

Dockerfile.yml编写：

FROM java:8-jre

ADD ./target/busdriver.jar /app/

CMD ["java", "-Xmx200m", "-jar", "/app/busdriver.jar"]

ENV TZ=Asia/Shanghai

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

EXPOSE 8080

## DockerCompose安装

1. 首先检查linux有没有安装python-pip包，直接执行 yum install python-pip
2. 没有python-pip包就执行命令 yum -y install epel-release
3. 执行成功之后，再次执行yum install python-pip
4. 对安装好的pip进行升级 pip install --upgrade pip
5. pip install docker-compose

如果报错

pip install --upgrade backports.ssl\_match\_hostname

1. docker-compose --version

## DockerCompose编排微服务

1. 启动容器docker-compose up
2. 停止容器docker-compose down
3. docker-compose.yml编写

version: '2'

services:

gateway:

environment:

registryIP: $registryIP

image: gateway

restart: always

ports:

- 17900:17900

locationserver:

environment:

registryIP: $registryIP

image: locationserver

restart: always

## 配置Jenkins自动更新代码

具体使用请看https://www.jianshu.com/p/b524b151d35f

# 贵交科服务架构设计说明

根据贵交科的实际情况，对微服务的设计做如下说明。使用zuul做服务网关，同时把zuul做一个集群，在zuul服务网关之前使用nginx做一个负载均衡地址统一对外。同时在zuul网关上面也做负载均衡和授权认证。使用eureka做服务的注册和服务器发现功能，eureka也做成集群化的东西。以便于实现高可用的注册中心。使用springcloudconfig做所有配置文件的管理操作，同时也做成集群化的形式实现高可用的配置文件管理中心。普通服务做成微服务集群，具体做法不详细介绍，参考本文的以上章节就能完成。

代码仓库和配置文件仓库都是用自己搭建的私有gitlab服务器上边，实现代码上传自动编译，自动部署的效果。

