# 复习

## 消息队列的作用

项目做成分布式的，系统中web服务器，有微服务提供者服务器，微服务提供者停机了，web程序出异常。数据没有被保存。Web程序把数据发到消息队列，微服务提供者从消息队列中取数据。保证数据会被处理。提高了程序可靠性。

流量削峰

## 消息队列与redis,mysql对比

Mysql把数据保存到硬盘上，mysql支持sql语句，有统计功能，支持事务。

请求多了，查询mysql速度慢，提高查询速度，把经常要查询的数据放在redis中。

为了提高程序可靠性用消息队列。解决高并发。数据进来，马上被取走。

## 消息队列5种模式

1. 简单模式：一个消费者
2. 工作模式：多个消费者，每个消费取到的数据是不一样的。
3. 订阅模式：每个消费者取到数据是一样的。
4. 路由模式：每个消费者取到数据是不一样的。发数据，数据有routingKey,消息队列服务器根据routingKey把数据放到不同的队列中。
5. 主题模式：把一类数据放到一个队列中，消费者1绑定队列1，队列1绑定到交换机，绑定时rountingKey是book.\*.提供发的数据的routingKey是book.order

# 拼多整合rabbitMQ

## 需求

由于订单业务逻辑相对复杂.同时入库3张表.当请求很多时，操作数据库很慢，有可能超时。所以使用消息队列实现订单入库.

角色划分:

消息提供者:pd-web

消息消费者:pd-orderConsumer

项目存在的问题，如图-93 所示

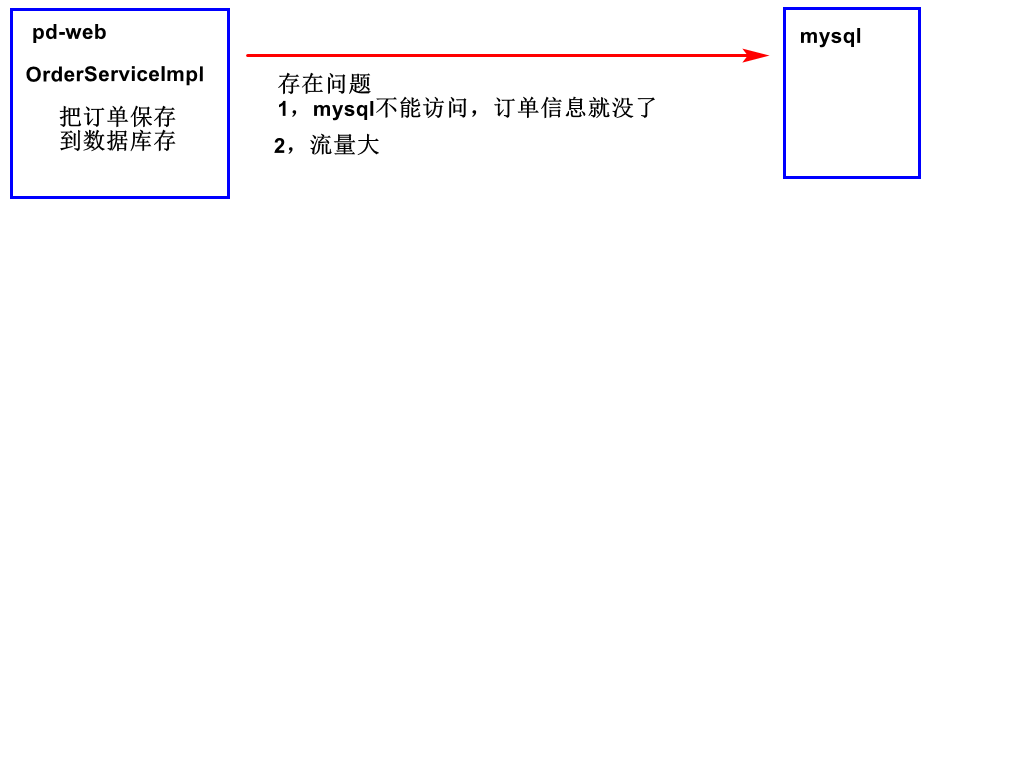


图- 93

为解决这个问题，一般需要在项目中加入消息队列, 如图-94 所示。

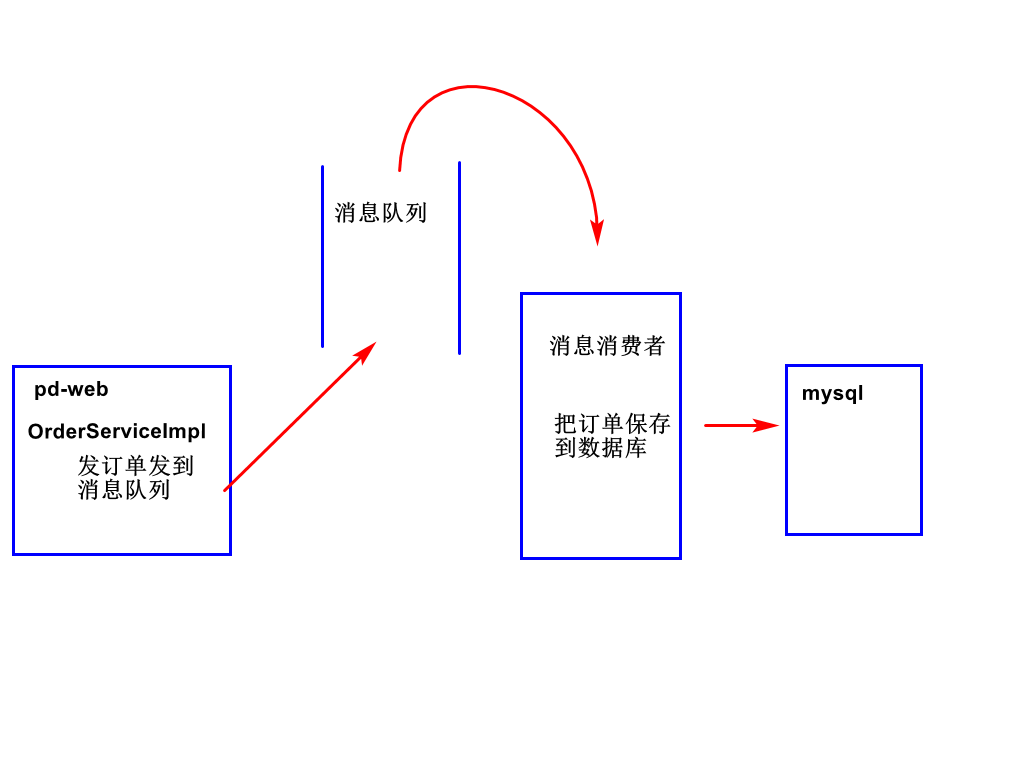


图- 94

## Pd-web提供消息

### 添加依赖

Pd-web中添加依赖。

<!-- rabbitmq相关依赖 -->

<dependency>

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

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

</dependency>

### 添加配置文件

拷贝下面内容到pd-web/src/main/resources/application.properties文件中。内容如下，修改消息队列服务器ip地址。

spring.rabbitmq.host=192.168.216.230

spring.rabbitmq.port=5672

spring.rabbitmq.virtualHost=/pd

spring.rabbitmq.username=pdadmin

spring.rabbitmq.password=pdadmin

### 设置队列

在com.pd包下创建SendConfig类

**package** com.pd;

**import** org.springframework.amqp.core.Queue;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

//消息提供者，用的是简单模式，配置队列名称

//配置提供者信息，在项目启动时，为SendConfig创建对象

@Configuration

**public** **class** SendConfig {

//连接rabbitmq的服务器的amqp框架，需要Queue对象

//提供对象给amqp框架有两种方式

//1,application-context.xml中增加<bean>

//2,@bean注解

//queue是amqp.core包中的

@Bean

**public** Queue queue()

{

//orderQueue在rabbitmq服务器上不存在

//存在，属性不一致，就会出异常

Queue queue=**new** Queue("orderQueue");

**return** queue;

}

}

### 发送测试消息

在com.pd.controller包下创建SendController类

@Controller

**public** **class** SendController {

//amqpTemplate是amqp框架提供的

//项目启动时 amqp框架创建AmqpTemplate对象，

//对象放到spring容器中

@Autowired

AmqpTemplate amqpTemplate;

@RequestMapping("/send")

@ResponseBody

**public** String send()

{

String routingKey="orderQueue";

amqpTemplate.convertAndSend

(routingKey, "pd");

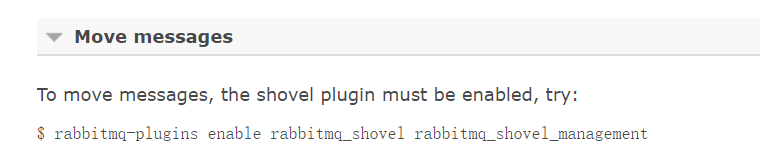
**return** "ok";

}

}

### 移出队列中数据

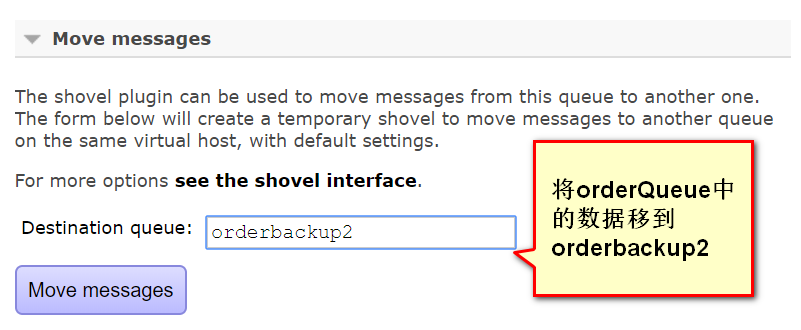
orderQueue要放订单对象，消费者收到订单后才能正确处理。需要把测试的字符串移走。



在xshell中执行下面命令安装插件

cd /usr/sbin

rabbitmq-plugins enable rabbitmq\_shovel rabbitmq\_shovel\_management



删除orderbackup2队列

### 拷贝pd-web为pd-orderConsumer

1. 把RunPdApp改名成RunOrderConsumer
2. 修改pom.xml中的artifactId

<artifactId>pd-orderConsumer</artifactId>

1. 修改application.yml中的端口号为81

server:

port: 81

### 发送order对象到消息队列

修改的是pd-web项目，不是pd-orderConsumer

修改com.pd.service.impl包中的OrderServiceImpl的saveOrder方法。

@Autowired

AmqpTemplate amqpTemplate;

**public** String saveOrder(PdOrder pdOrder) **throws** Exception {

String orderId = generateId();

pdOrder.setOrderId(orderId);

String routingKey="orderQueue";

amqpTemplate.convertAndSend(routingKey, pdOrder);

//删除数据库相关代码

**return** orderId;

}

发送数据后，在rabbitMq后台能查到OrderQueue中的数据, 如图- 102所示



图- 102

## pd-orderConsumer消费消息

### 创建监听类

在com.pd包下创建OrderConsumerListener

//消息的消费者

//amqp框架自动连上消息队列服务器

//取到消息后，amqp框架自动调用这个类

@Component

**public** **class** OrderConsumerListener {

@Autowired

OrderService orderService;

//amqp框架自动从orderQueue队列中取消息

//把收到字符串转成pdOrder对象

//取到消息后调用process

@RabbitListener(queues="orderQueue")

**public** **void** process(PdOrder pdOrder)

{

System.***out***.println("消费者");

System.***out***.println(pdOrder.toString());

**try** {

orderService.saveOrder(pdOrder);

} **catch** (Exception e) {

}

}

}

### 删除不相关代码

删除controller包

删除payment

删除不用的service

### 启动程序

在网上下订单后，orderConsumer收到消息后，把订单保存到数据库中。会发现数据库中的订单号与网页上的不一致。

### 修改OrderServiceImpl

从消息队列服务器取到的订单已经有id了，要注释掉下面两行。

//String orderId = generateId();

// pdOrder.setOrderId(orderId);

String orderId=pdOrder.getOrderId();

### 反序列化的bug

在saveOrder中收到pdOrder,addId属性值为空，解决方法是创建一个新虚拟主机，将消息发到新的虚拟主机中，再从新虚拟主机中取数据。

## 使用消息队列的优点

如果不使用消息队列，数据库访问超时或数据库关机，订单保存就会失败。使用了消息队列后，订单先存到消息队列服务器，pd-orderConsumer启动后，所有未处理订单都会被处理。

# Dubbo

Dubbo是一种微服务框架。

## 需求

11.11 用户请求很多，一台服务处理不过来，需要多台服务器。

京淘项目中使用springCloud实现了微服务。微服务就是把原先的service层代码独立出来。部署到一个新的服务器上,如图- 1所示。 微服务一般做成分布式。

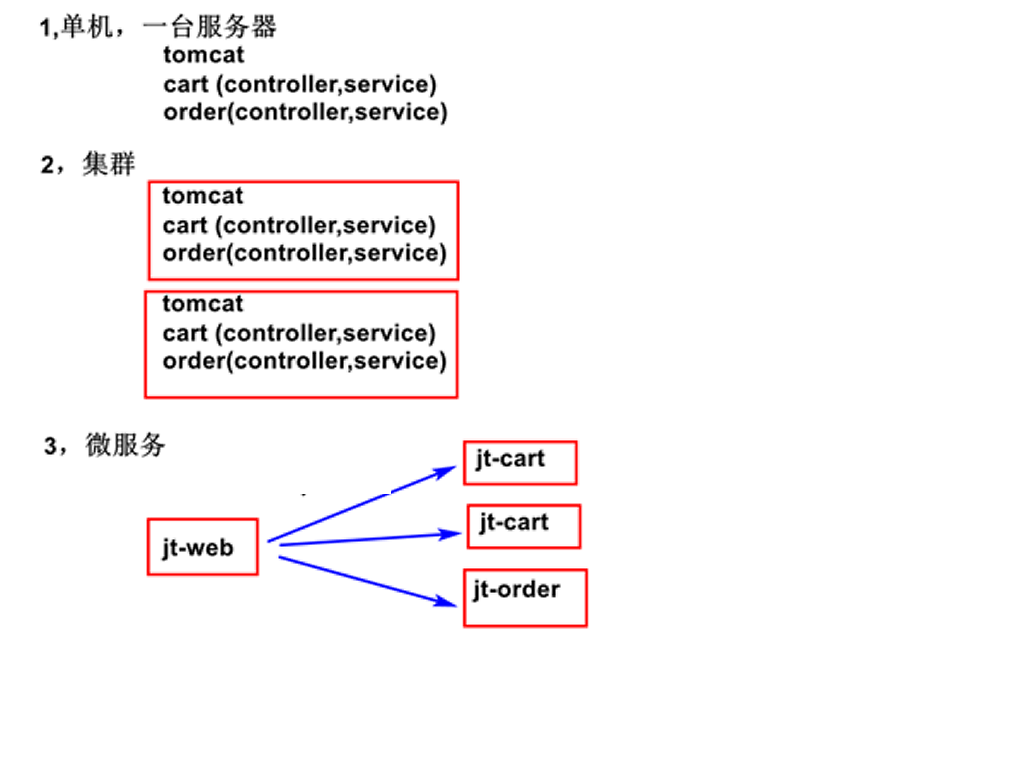


图- 1

## 分布式(微服务)优点

1. 处理更多的请求。
2. 故障隔离
3. 拆分成多个项目如cart,sso,order,web容易维护，好定位bug。
4. 可扩展，订单部署3台服务器，web访问集群nginx,nginx再访问3台订单服务器。
5. 单独发布。

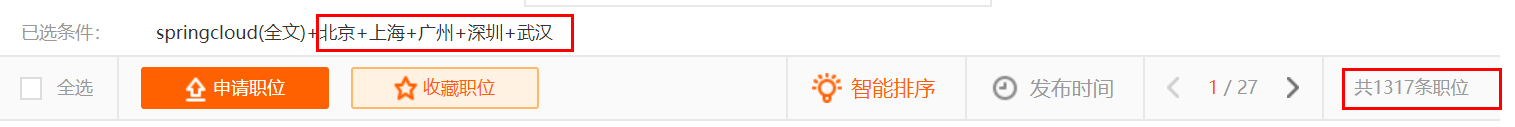
## 微服务实现

1. springcloud
2. dubbo
3. httpClient

### dubbo与springcloud对比

springCloud是新技术，dubbo是老技术，稳定

www.51job.com上springcloud招聘量

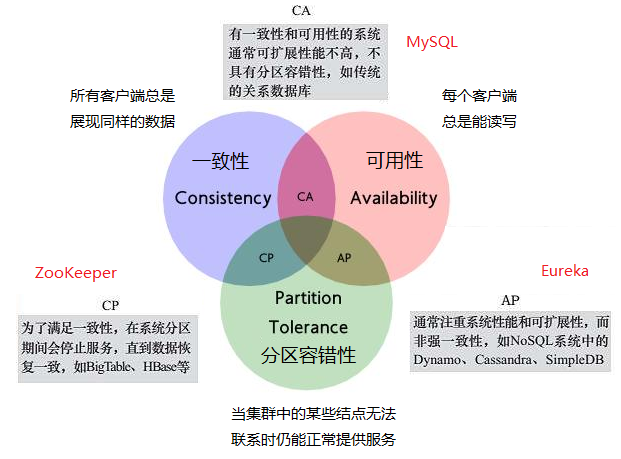


www.51job.com上dubbo招聘量



## CAP

### CAP介绍



CAP原则又称CAP定理，指的是在一个分布式系统中，Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性），三者不可兼得。它是分布式系统中最核心最重要的理论。

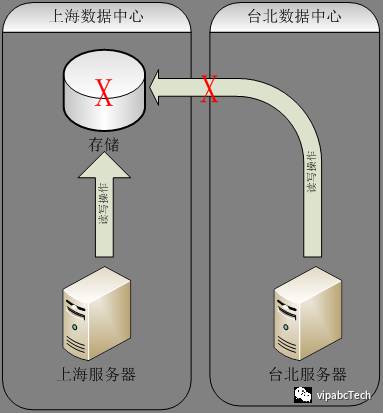
分布式系统的CAP理论：理论首先把分布式系统中的三个特性进行了如下归纳：

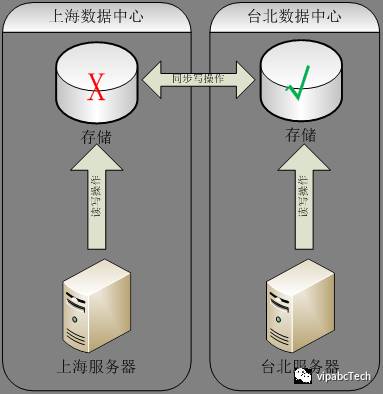
* 一致性（C）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）
* 可用性（A）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）
* 分区容错性（P）：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

CAP理论就是说在分布式系统中，最多只能实现上面的两点。而由于当前的网络硬件肯定会出现延迟丢包等问题，所以分区容忍性是我们必须需要实现的。所以我们只能在一致性和可用性之间进行权衡，要么选择CP要么选择AP，没有分布式系统能同时保证这三点。

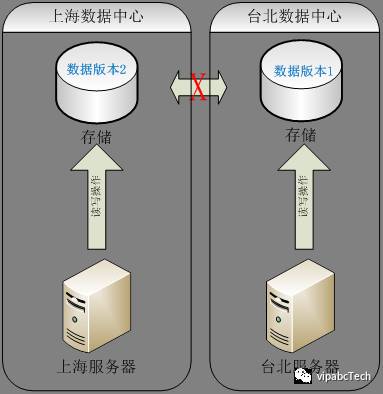
### Cap在数据库集群中的应用

#### P的实现





#### A的实现



台北将过期数据“版本1”返回给服务器实现了可用性。

由于返回的“版本1”与最新数据“版本2”不一致，违反了“一致性”

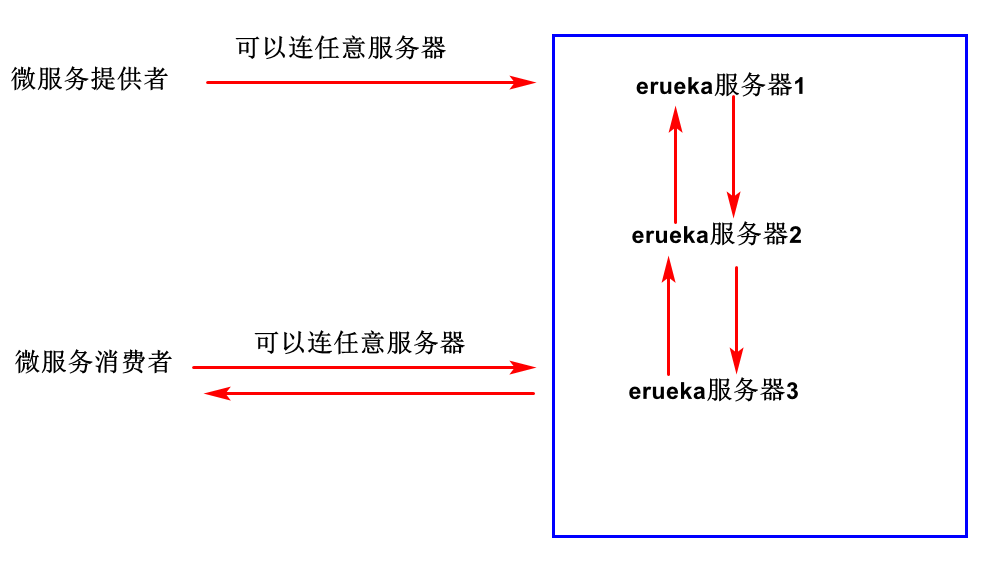
#### C的实现

等待网络恢复正常，上海存储将“版本2”同步至台北存储，台北存储将满足“一致性”的数据“版本2”返回给服务器。

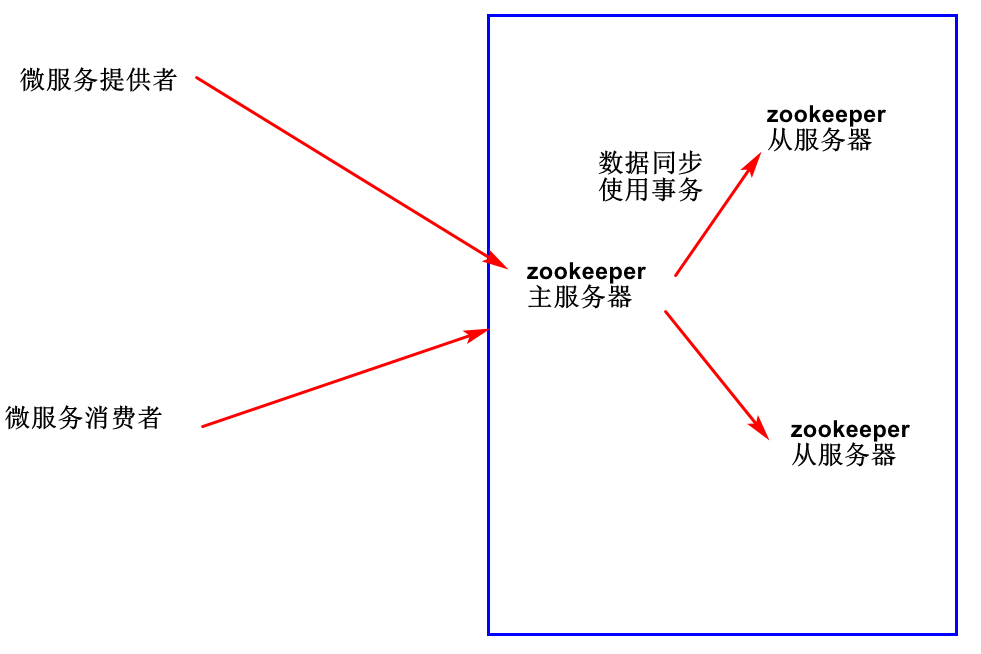
由于网络恢复时间的不确定性，请求可能会超时，违反“可用性”。

P必须实现。一般情况下 A比C优先。金融系统中C比A优先。

### CAP在zk和eureka中的实现



只要有一个eureka服务器可用，服务就可用，保证了可用性。实现了AP



数据同步使用事务实现了一致性。

主服务器岩机了，几个从服务器要选举主服务器，选举期间，zookeeper不可用，无法实现可用性。

Zk实现了CP

## 安装zookeeper

### Zookeeper是服务注册中心

可以管理tomcat,redis,dubbo

Zookeeper当作服务注册中心

服务可以是tomcat服务，redis服务，solr服务，也可以是dubbo微服务

不用服务注册中心。是通过ip,端口号直接连接服务器

在企业把tomcat,mysql,redis,solr的ip，端口号发给zookeeper,客户端从zookeeper上取服务器的ip,端口号，当服务器的ip,端口号发生变化，客户端代码不变

用dubbo做微服务，注册中心用zookeeper

用springcloud,注册中心用eureka

微服务的注册中心用eureka更合适。

### 与eureka区别

zookeeper是开源,可以管理tomcat,redis,dubbo，solr等服务.

Eureka 2018年6月29日开始不再开源，适合做微服务的注册中心。

### zookeeper的作用

zookeeper的作用如图-3 所示

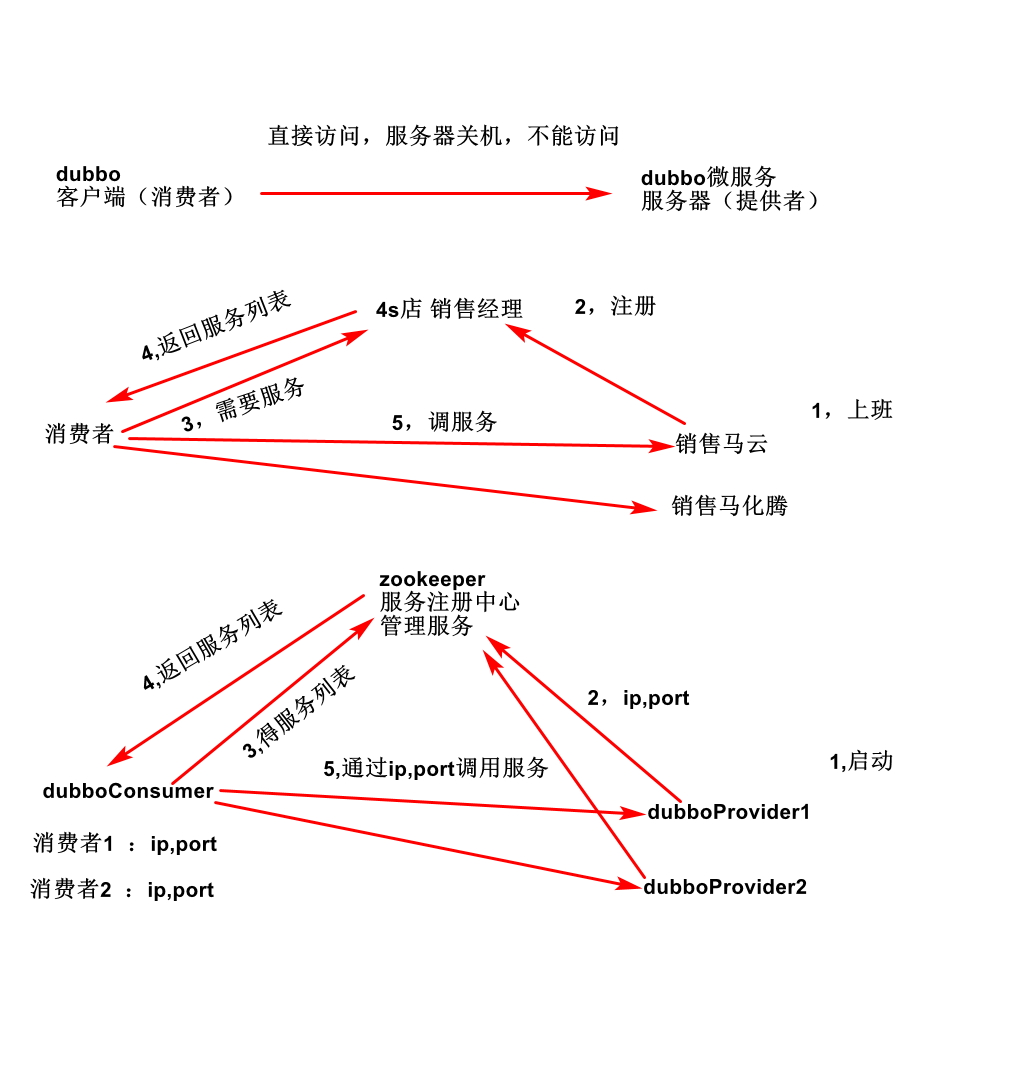


图- 3

客户端直连服务提供者，无法实现集群。

ZooKeeper是一个分布式应用程序协调服务。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

当客户端发起请求时,zookeeper返回正确的服务器地址.

业务逻辑说明:

1. 当服务的提供者启动时,会将服务的名称:IP:端口会写入zookeeper.
2. zookeeper内部会维护服务列表,某个服务提供者关机了，服务还能正常进行
3. 当消费者需要访问服务时,需要先访问zookeeper获取服务列表信息.之后将服务列表保存到本地缓存中.方便后续的访问.在客户端内部有负载均衡的算法,筛选出一台服务器,之后进行访问.
4. 如果zookeeper服务器出现宕机现象.这时注册中心通过心跳检测的方式判断服务器是否宕机.如果服务器宕机则会将该服务器信息从服务列表中删除.之后将新的服务列表发送消费者(客户端)进行更新.

### Zookeeper下载

下载网址如图-5 所示。

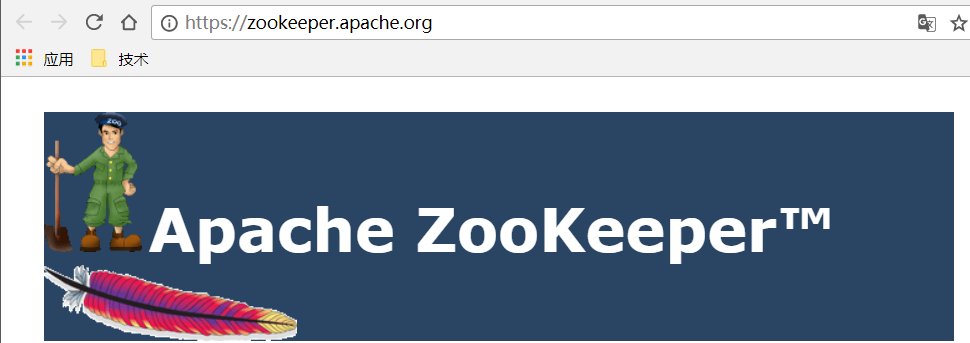


图- 5

### 上传安装文件

Centos6.5

不用在/usr/local/src中创建新文件夹。

上传“\亿发,移动端课前资料\dubbo\zookeeper安装包”文件夹中的 zookeeper-3.4.8.tar.gz到/usr/local/src

执行下面的命令解压zookeeper

cd /usr/local/src

tar -xzvf zookeeper-3.4.8.tar.gz

### 修改配置文件

1.在zookeeper-3.4.8目录下创建文件夹data和log

cd /usr/local/src

cd zookeeper-3.4.8

mkdir data log

2.使用fz下载 zookeeper3.4.8/conf/zoo\_sample.cfg

3. 去掉文件名中的\_sample,修改文件名为zoo.cfg

4.修改zoo.cfg文件中12行的dataDir的值，增加dataLogDir。

dataDir=/usr/local/src/zookeeper-3.4.8/data

dataLogDir=/usr/local/src/zookeeper-3.4.8/log

修改后如图-15 所示

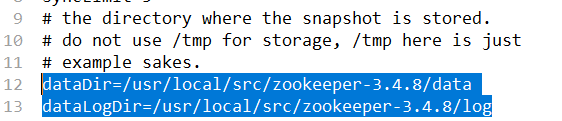


图- 15

6.使用fz上传zoo.cfg到zookeeper3.4.8/conf中

### 启动zk服务器

cd /usr/local/src/zookeeper-3.4.8/bin/

sh zkServer.sh start

启动成功后显示started，如图-16 所示

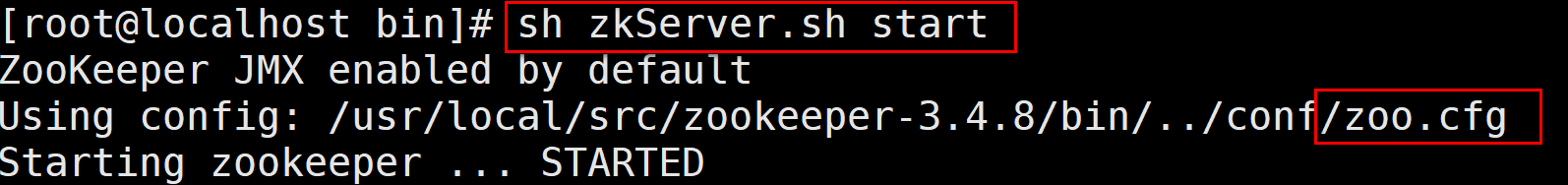


图- 16

### Zk客户端的使用

//进入zookeeper客户端

[root@localhost bin]# sh zkCli.sh

//列出注册的服务

[zk: localhost:2181(CONNECTED) 0] ls /

[zookeeper]

//退出

[zk: localhost:2181(CONNECTED) 1] quit

## Dubbo监控系统

Dubbo监控系统用来显示服务信息如图-17 所示。

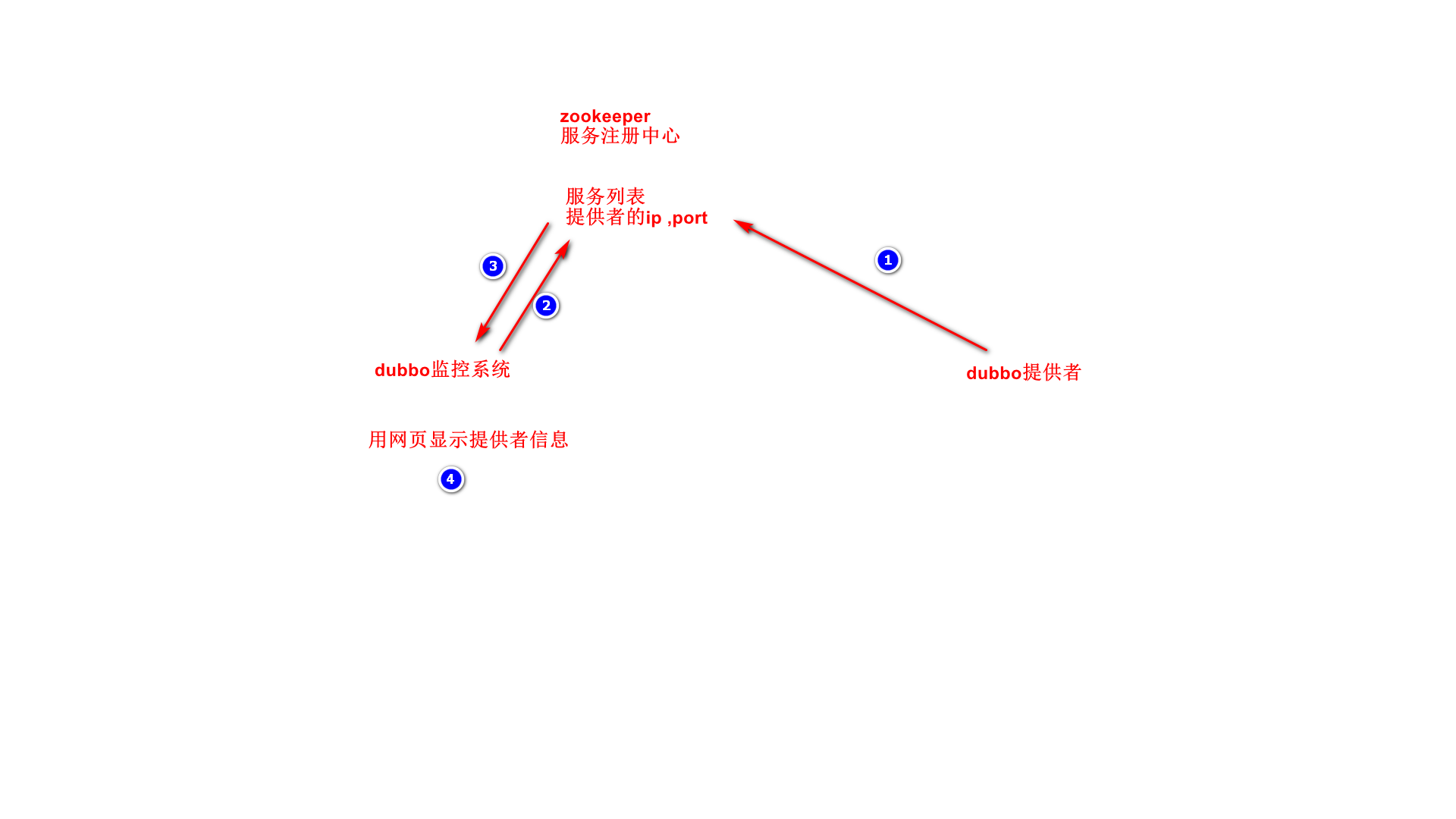


图- 17

在windows中部署dubbo监控系统。通过监控系统查看dubbo的提供者信息。

有的电脑dubbo监控部署后，不能正常访问，不影响后面的学习，我们可以使用zk客户端查看服务信息。

### 部署tomcat

1. Windows中拷贝\亿发,移动端课前资料\dubbo\dubbo控制台\tomcat-dubbo文件夹到新文件夹中。
2. 在windows查看jdk版本，如图- 18所示.

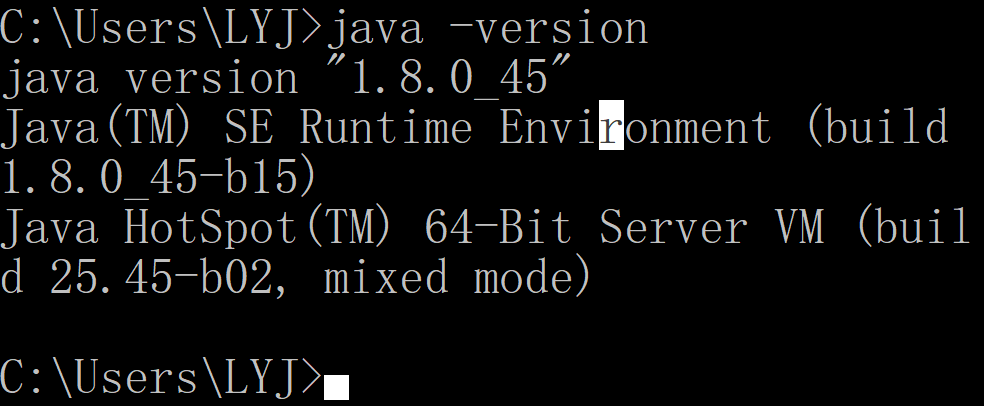


图- 18

1. 根据jdk版本拷贝对应的root.war到webapps中
2. 双击\tomcat-dubbo\bin\startup.bat启动tomcat
3. Root.war会自动解压
4. 第一次启动，还没有修改zookeeper服务器ip地址，会报下面的错误如图- 19所示。

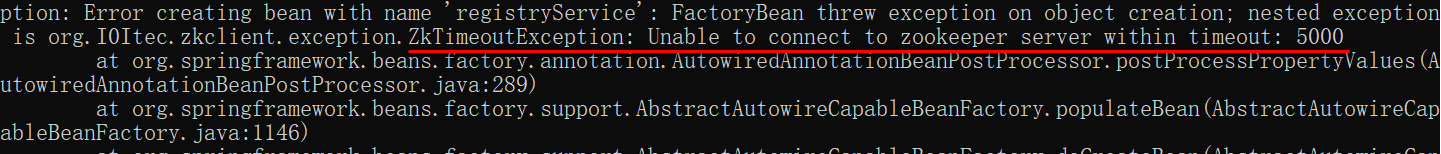


图- 19

### 修改dubbo配置文件

1.将\ROOT\WEB-INF\dubbo.properties中的ip地址改为虚拟机的IP地址，端口号2181不用改，要保留下来。修改后记得保存，如图-20 所示.

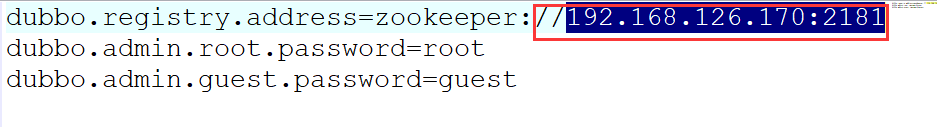


图- 20

1. 关闭tomcat的dos窗口, 双击\tomcat-dubbo\bin\startup.bat重启tomcat
2. 效果展现

浏览器中输入<http://localhost:8090>

用户名是root 密码是root

登录成功后如图-21 所示



图- 21

改成中文，查看提供者，无提供者。

## Dubbo案例

### Dubbo原理

Dubbo运行过程如图- 22所示

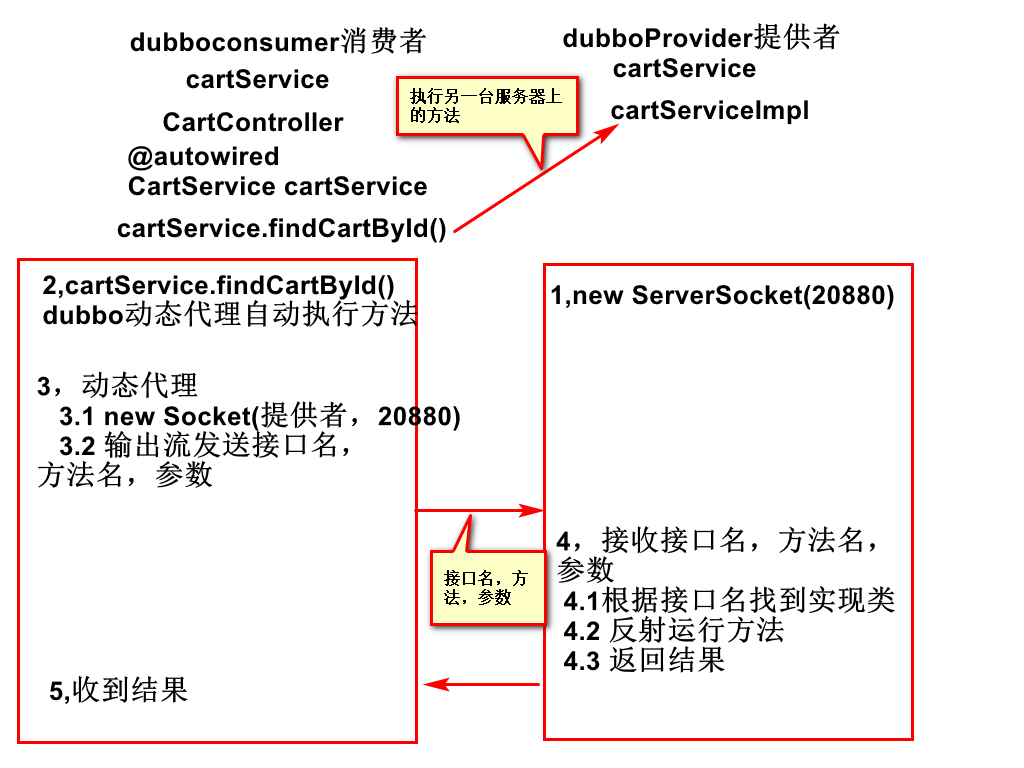


图- 22

像调用本地方法一样去调用远程服务器上的方法。

### 项目分析

项目之间关系如图- 23所示

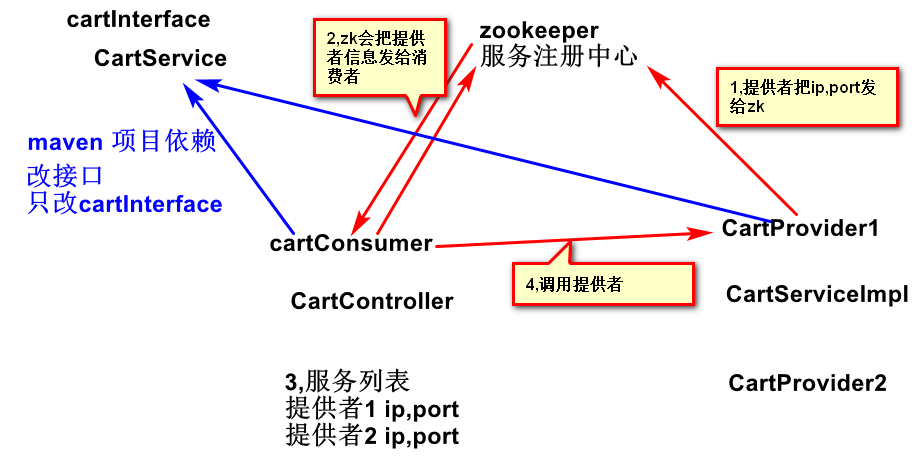


图- 23

提供者可以理解成服务器

消费者可以理解成客户端

### 创建dubboInterface项目

项目类型是quick start

groupId是com.tedu

artifactId是dubboInterface

1. 创建包com.tedu.service，如图-24 所示

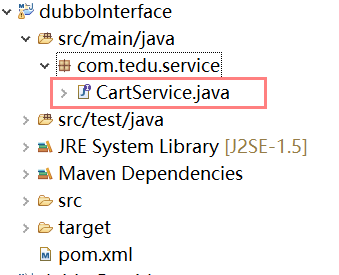


图- 24

1. 创建接口

**public** **interface** CartService {

**public** String findCartByUserId(Long userId);

}

1. 发布jar

Run as 🡪maven install

### 创建第一个提供者 dubboProvider1

1. 创建maven项目，类型是 quickstart，项目名为dubboProvider1
2. 依赖cartInterface项目。
3. 添加dubbo，zkclient依赖

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.8.4</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

添加dubbo依赖，因为dubbo依赖spring等jar,这些jar会自动添加进来，如图-25 所示

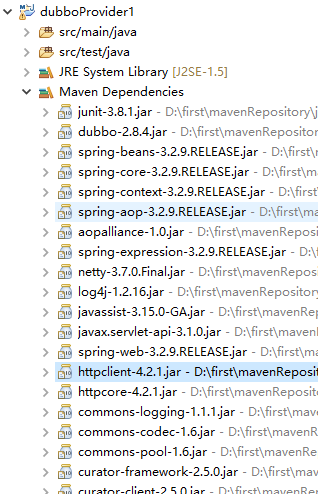


图- 25

#### 添加依赖出错解决方法

1.添加依赖出错第一种解决方法：强制更新，如图- 26所示

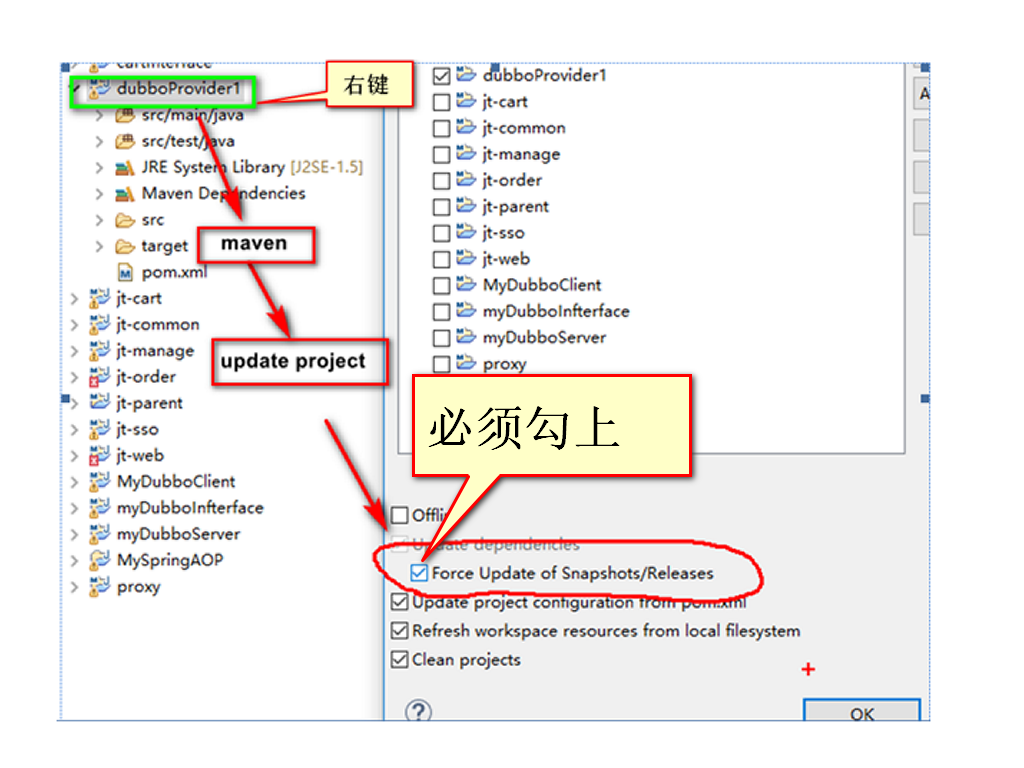


图- 26

2.添加依赖出错第二种解决方法：拷贝别人maven的本地仓库

通过eclipse🡪preferences查看本地仓库的位置，如图- 27所示

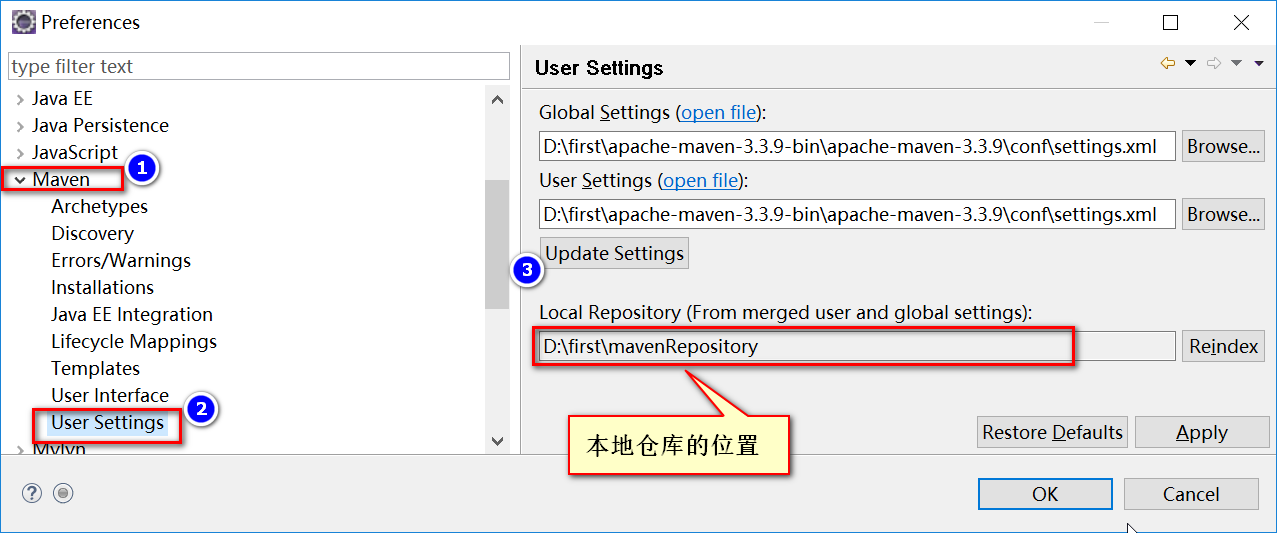


图- 27

在拷贝别人或老师网盘的maven本地仓库之前，先备份自己的仓库。

拷贝完后，本地仓库文件夹应该有org等文件夹如图-28 所示

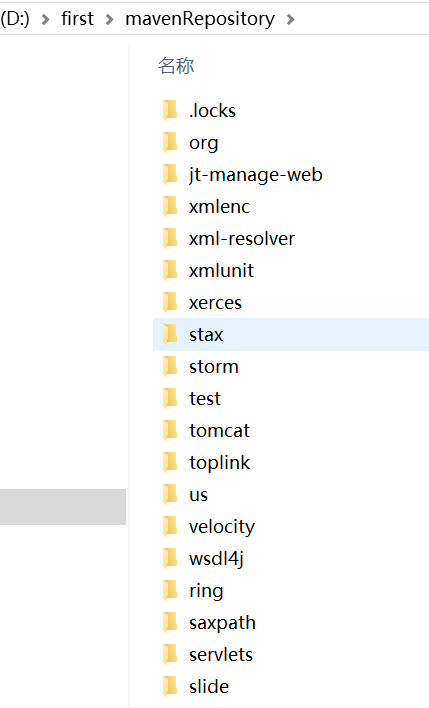


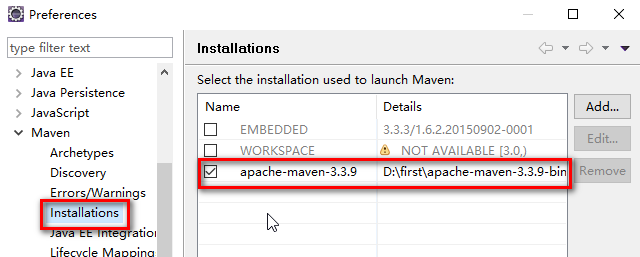
图- 28

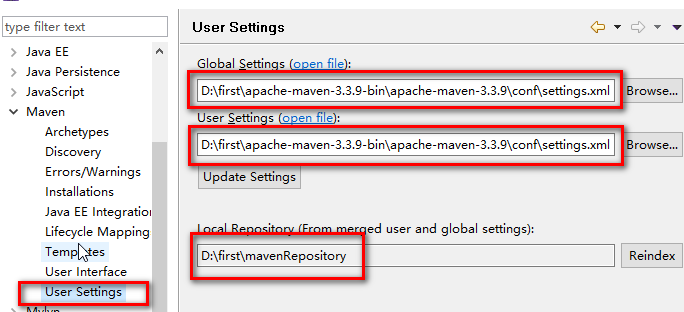
3.添加依赖出错第三种解决方法：更改工作区

尽量不要使用此种方法，因为改了工作区，要重新配置maven和jdk

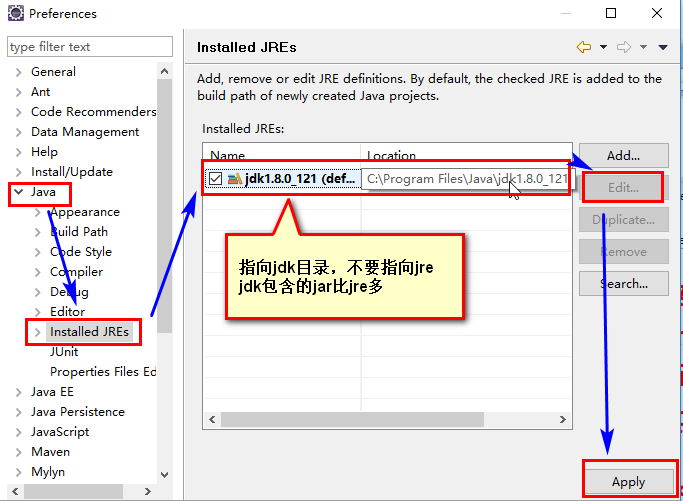
Eclipse🡪file🡪switch workspace

更改工作区后，要重新设置maven





重新设置jdk



1. 添加服务实现类

创建包com.tedu.service

**public** **class** CartServiceImpl **implements** CartService {

**public** String findCartByUserId(Long userId) {

**return** "第1个提供者 cart userId=" + userId;

}

}

1. 添加约束文件

查看jar所在位置如图-29 所示

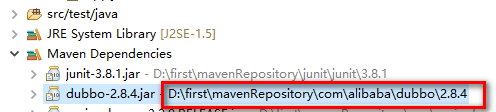
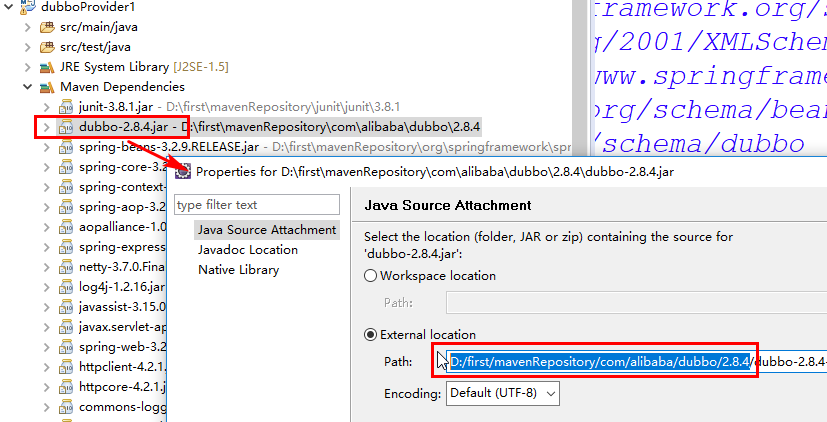


图- 29



解压dubbo-2.8.4.jar，从meta-inf中找到dubbo.xsd

Eclipse中Window🡪preference🡪xml🡪xml catalog添加catalog，如图- 30所示

Location指向本机dubbo.xsd文件, Key是[*http://code.alibabatech.com/schema/dubbo/dubbo.xsd*](http://code.alibabatech.com/schema/dubbo/dubbo.xsd),key是从xml schemaLocation拷贝过来的。

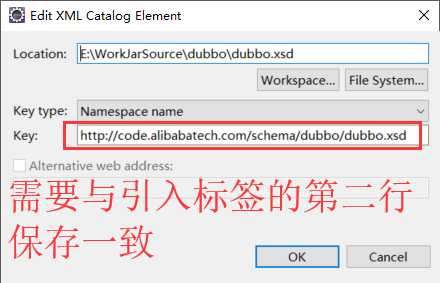


图- 30

1. 在src\main\java下创建applicationContext-provider.xml,不要放在包中

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*

*http://code.alibabatech.com/schema/dubbo*

*http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 微服务提供者配置文件，作用是把提供者的端口，

实现类信息发给服务注册中心zookeeper -->

<!-- 1,设置应用（微服务）名称 -->

<dubbo:application name=*"cart"*/>

<!-- 2,设置zookeeper的ip地址和端口号 -->

<dubbo:registry timeout=*"900000"*

address=*"zookeeper://192.168.216.210:2181"*>

</dubbo:registry>

<!-- 3,配置端口号，提供者通过端口号连上来 -->

<!-- name设置的是协议名称可以是http -->

<dubbo:protocol port=*"20880"* name=*"dubbo"*>

</dubbo:protocol>

<!-- 4,配置实现类，dubbo框架调用实现类 -->

<bean id=*"CartServiceImpl"*

class=*"com.tedu.cart.service.CartServiceImpl"*>

</bean>

<!-- 5,配置对外提供的服务 -->

<!-- 消费者在一台服务器，通过网络访问提供者，传过来接口名

提供者根据当前配置找到实现类，通过反射运行方法。

-->

<dubbo:service timeout=*"900000"*

interface=*"com.tedu.cart.service.CartService"*

ref=*"CartServiceImpl"*

>

</dubbo:service>

</beans>

1. main启动项目

**public** **class** provider1Main {

**public** **static** **void** main(String[] args) **throws** Exception {

ClassPathXmlApplicationContext context=

**new** ClassPathXmlApplicationContext("applicationContext-provider.xml");

context.start();

System.***out***.println("提供者1启动");

while(true)

{

}

}

}

1. zkCli查看服务

cd /usr/local/src/zookeeper-3.4.8/bin

sh zkCli.sh

[zk: localhost:2181(CONNECTED) 1] ls /

[dubbo, zookeeper]

//查看 dubbo服务

[zk: localhost:2181(CONNECTED) 2] ls /dubbo

[com.tedu.service.CartService]

//查看cartService有没有提供者

[zk:localhost:2181(CONNECTED)3] ls /dubbo/com.tedu.service.CartService

//查看cartService有多少个提供者

[zk:localhost:2181(CONNECTED)3] ls /dubbo/com.tedu.service.CartService/providers

//退出zkCli

[zk:localhost:2181(CONNECTED)3] quit

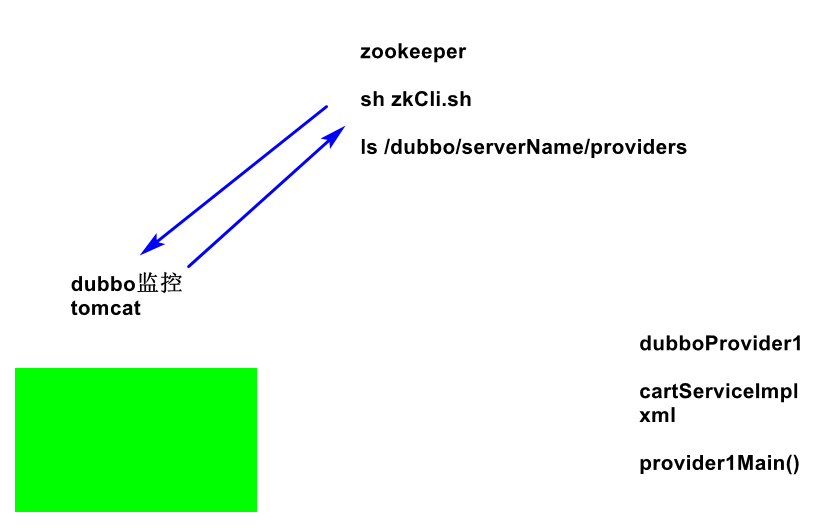
1. dubbo监控中能查到服务,选中服务查看详细信息

访问localhost:8090，如图- 31所示



图- 31

1. dubbo监控与zookeeper的关系



1. console中退出，监控中看不到服务

### 创建第二个提供者

1. 拷贝dubboProvider1项目，改名成dubboProvider2
2. 修改dubboProvider2的pom.xml
3. 修改cartServiceImpl中的字符串为“第2个提供者”

**public** String findCartByUserId(Long userId) {

**return** "第2个提供者 cart userId=" + userId;

}

1. 修改dubbo端口号,配置文件修改后如下

<!-- 1. 设置应用名称-->

<dubbo:application name=*"cart"*/>

<!-- 2.配置zookeeper地址 -->

<dubbo:registry address=*"zookeeper://192.168.216.202:2181"*>

</dubbo:registry>

<!-- 3.配置服务的端口号 -->

<dubbo:protocol port=*"20889"* name=*"dubbo"*>

</dubbo:protocol>

<!-- 4.配置实现类的类名 -->

<bean class=*"com.tedu.dubboProvider.CartServiceImpl"* id=*"cartService"*>

</bean>

<!-- 5.配置接口名,开放服务 -->

<dubbo:service interface=*"com.tedu.service.CartService"* ref=*"cartService"*>

</dubbo:service>

1. 修改provider2Main中的字符串

**public** **class** provider2Main {

**public** **static** **void** main(String[] args) **throws** Exception {

ClassPathXmlApplicationContext context=

**new** ClassPathXmlApplicationContext("applicationContext-provider.xml");

context.start();

System.***out***.println("提供者2启动");

while(true)

{

}

}

}

1. 启动服务提供者2，向zookeeper注册服务
2. Dubbo 监控中查看提供者2
3. 多个提供者形者集群。

### 创建消费者consumer

1. 创建类型为 quickstart的maven项目dubboConsumer
2. 依赖dubboInterface项目
3. 依赖dubbo,zkClinet框架

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.8.4</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

1. 配置applicationContext-consumer.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://code.alibabatech.com/schema/dubbo*

*http://code.alibabatech.com/schema/dubbo/dubbo.xsd "*>

<!--1，设置应用名-->

<dubbo:application name=*"cart"* />

<!--2，设置注册中心地址 -->

<dubbo:registry

timeout=*"50000"*

address=*"zookeeper://192.168.216.230:2181"* />

<!-- controller

@autowired spring从本项目找到一个实现类

LoginService loginService

@autowired

CartService cartService

spring框架发现CartService在配置文件中配置过

spring框架不找对象，交给dubbo框架

dubbo框架运行动态代理，用socket联接提供者，

把接口名，方法名，参数发给提供者

提供者用反射运行方法，把结果通过socket返回来

check=false 消费者启动时，不去检查提供者是否启动

-->

<dubbo:reference

timeout=*"50000"*

check=*"false"*

id=*"cartService"*

interface=*"com.tedu.service.CartService"* />

</beans>

1. 启动消费者

**public** **class** CartController {

**public** **static** **void** main(String[] args) **throws** Exception {

ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext-consumer.xml");

//通过id查找对象

CartService cartService = (CartService) context.getBean("cartService");

**while** (**true**) {

String string = cartService.findCartByUserId(18L);

System.***out***.println("服务消费者收到的结果:" + string);

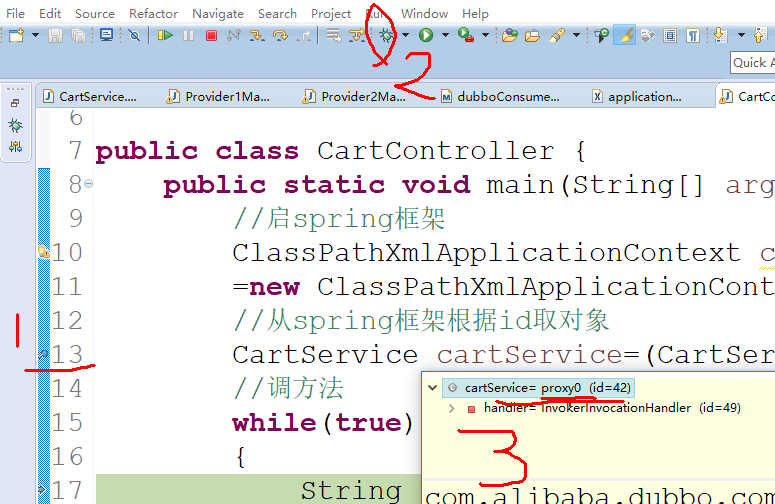
Thread.*currentThread*().*sleep*(1000);

}

}

}

两个服务提供者都会被调用，随机分配



1. 通过zkCli查看信息

cd /usr/local/src/zookeeper-3.4.8/bin

sh zkCli.sh

[zk: localhost:2181(CONNECTED) 1] ls /

[dubbo, zookeeper]

//查看 dubbo服务

[zk: localhost:2181(CONNECTED) 2] ls /dubbo

[com.tedu.service.CartService]

//查看cartService有多少个提供者

[zk:localhost:2181(CONNECTED)3] ls /dubbo/com.tedu.service.CartService/consumers

//退出zkCli

[zk:localhost:2181(CONNECTED)3] quit

1. 倍权提供者
2. 禁用提供者
3. 启用提供者
4. 关闭zookeeper

cd /usr/local/src/zookeeper-3.4.8/bin

sh zkServer.sh stop

Dubbo客户端有缓存。能继续正常运行

1. 关闭provider1
2. 关闭provider2

### 总结

# 总结

