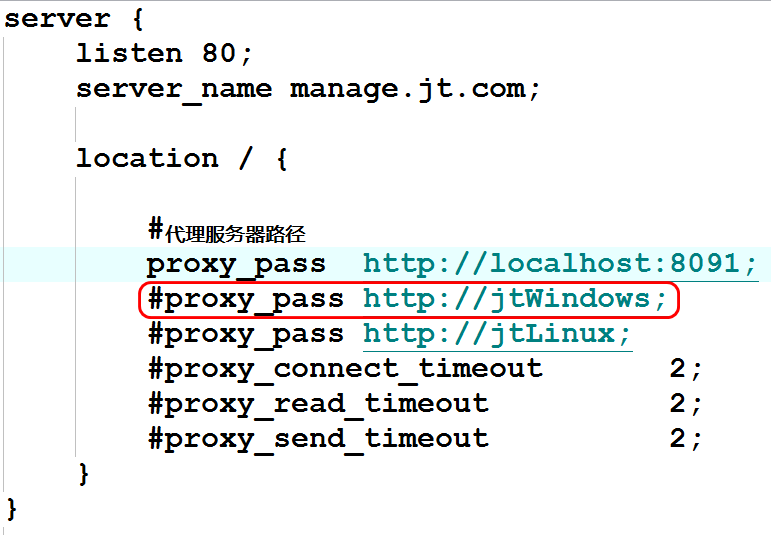
# 报错检查

## 代码报错异常

### 空指针异常

由于配置了轮询,导致报错



### json转化异常

问题说明:由于不同的数据使用了相同的key导致json转化异常.

com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException: Unrecognized field "status" (class com.jt.pojo.ItemDesc), not marked as ignorable (4 known properties: "created", "updated", "itemDesc", "itemId"])

at [Source: (String)"{"status":201,"msg":"调用失败!!!","data":null}"; line: 1, column: 14] (through reference chain: com.jt.pojo.ItemDesc["status"])

at com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException.from(UnrecognizedPropertyException.java:61)

at com.fasterxml.jackson.databind.DeserializationContext.handleUnknownProperty(DeserializationContext.java:823)

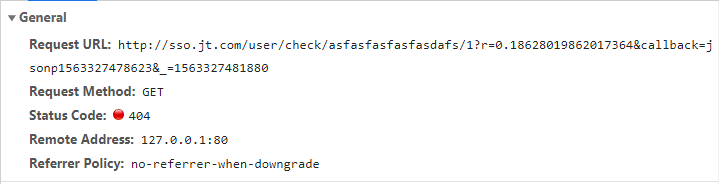
at com.fasterxml.jackson.databind.deser.std.StdDeserializer.handleUnknownProperty(StdDeserializer.java:1153)

## 实现用户校验

### 业务接口定义

|  |  |
| --- | --- |
| 请求方法 | GET |
| URL | **http://sso.jt.com/user/check/{param}/{type}** |
| 参数 | 格式如：chenchen/1  其中chenchen是校验的数据  Type为类型，可选参数1 username、2 phone、3 email |
| 示例 | http://sso.jt.com/user/check/chenchen/1 |
| 返回值 | {  status: 200 //200 成功，201 没有查到  msg: “OK” //返回信息消息  data: false //返回数据true用户已存在，false用户不存在，可以  } |
|  |  |

### url地址



### 编辑Controller

@RestController

@RequestMapping("/user")

**public** **class** UserController {

@Autowired

**private** UserService userService;

/\*\*

\* 经过跨域请求 返回系统数据 data:true(已存在)/false(不存在)

\* http://sso.jt.com/user/check/asfasfasfasfasdafs/1?r=0.18628019862017364&callback=jsonp1563327478623&\_=1563327481880

\*/

@RequestMapping("/check/{param}/{type}")

**public** JSONPObject findCheckUser(

String callback,

@PathVariable String param,

@PathVariable Integer type) {

JSONPObject jsonpObject = **null**;

**try** {

//查询数据库,检查数据是否存在

**boolean** flag = userService.findUserCheck(param,type);

jsonpObject = **new** JSONPObject(callback,SysResult.*success*(flag));

} **catch** (Exception e) {

e.printStackTrace();

jsonpObject = **new** JSONPObject(callback,SysResult.*fail*());

}

**return** jsonpObject;

}

}

### 编辑Service

@Service

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** UserMapper userMapper;

/\*\*

\* true 当前用户输入内容 已存在

\* false 表示数据可以使用

\* param:用户输入的数据

\* type: 参数类型 1 username 2 phone 3 email

\*/

@Override

**public** **boolean** findUserCheck(String param, Integer type) {

//1.定义查询的字段

String column =

(type==1)?"username":

((type==2)?"phone":"email");

//2.校验数据库中是否有数据???

QueryWrapper<User> queryWrapper = **new** QueryWrapper();

queryWrapper.eq(column, param);

**int** count = userMapper.selectCount(queryWrapper);

**return** count==0?**false**:**true**;

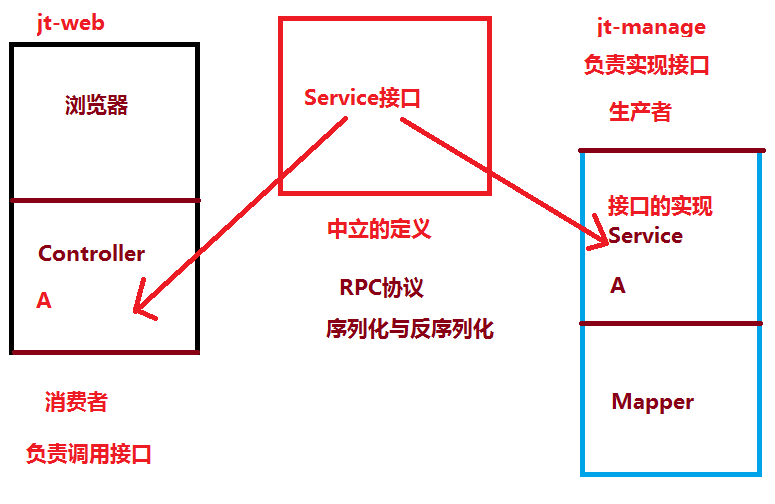
}

}

# Dubbo框架

## SOA思想

### 代码简化策略



### SOA思想

**面向服务**的架构（SOA）是一个组件模型，**它将应用程序的不同功能**单元（称为服务）进行拆分，并通过这些服务之间**定义良好的接口**和契约联系起来。**接口是采用中立的方式进行定义的**，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种各样的系统中的服务可以以一种统一和通用的方式进行交互。

总结:服务与服务之间通过接口的方式进行交互.不考虑中间环节.

### RPC协议

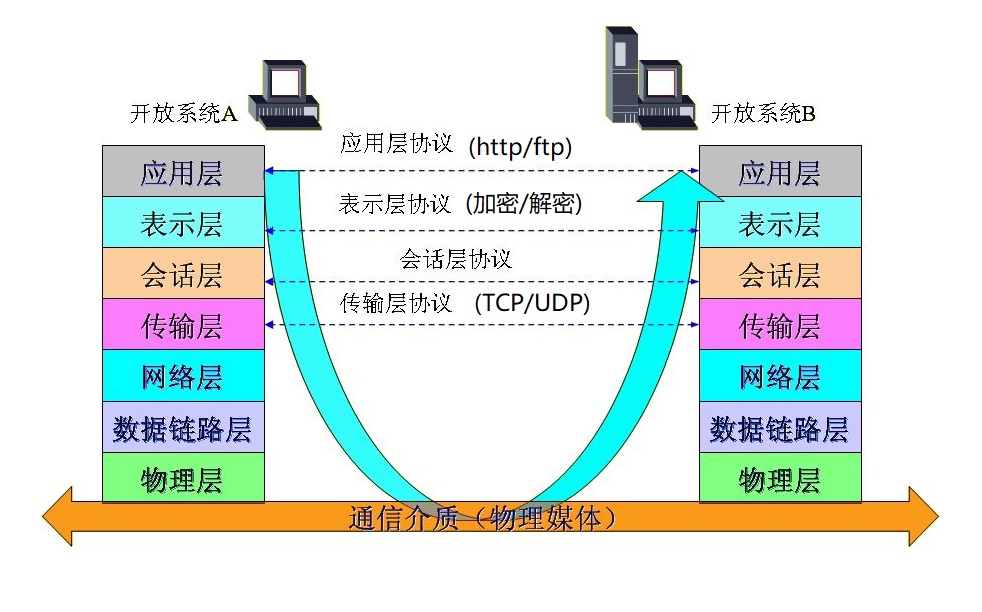
RPC（Remote Procedure Call）—[远程过程调用](https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8/7854346)，它是一种通过[网络](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C/143243)从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。[RPC协议](https://baike.baidu.com/item/RPC%E5%8D%8F%E8%AE%AE)假定某些[传输协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE/8048821)的存在，如TCP或UDP，为通信程序之间携带信息数据。在**OSI**[网络通信](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1/9636548)模型中，RPC跨越了[**传输层**](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82/4329536)**和**[**应用层**](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E5%B1%82/4329788)。RPC使得开发包括网络[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F)多程序在内的应用程序更加容易。

总结:RPC封装了TCP/UDP可以实现远程数据调用.

### RPC与HTTP区别

网络7层协议如图所示.

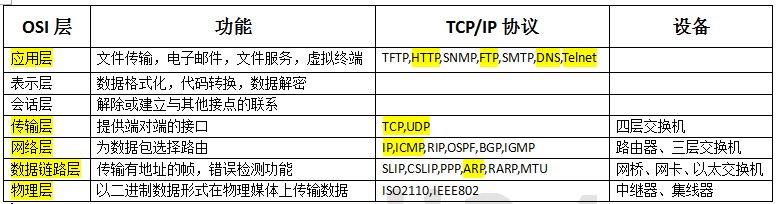
如图-22所示



**图-**22

层级关系与对应的协议.

如图-23所示



**图-**23

区别:

1.RPC是传输层协议(4层).而HTTP协议是应用层协议(7层).

2.RPC协议可以直接调用中立接口,HTTP协议不可以.String

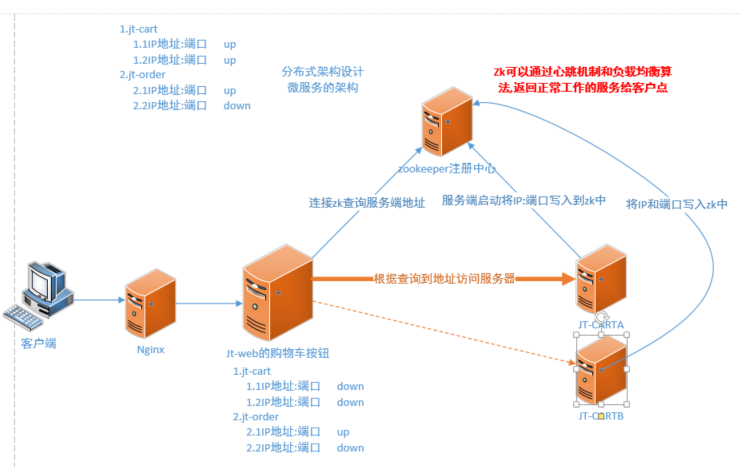
3.RPC通信协议是长链接,HTTP协议一般采用短连接需要3次握手(可以配置长链接添加请求头Keep-Alive: timeout=20).

(长连接，指在一个连接上可以连续发送多个[数据包](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%8C%85/489739)，在连接保持期间，如果没有数据包发送，需要双方发链路检测包。)

4.RPC协议传递数据是加密压缩传输.HTTP协议需要传递大量的请求头信息.

5.RPC协议一般都有**注册中心**.有丰富的监控机制.

### 注册中心



步骤:

1. 当生产者启动时,将会服务的名称:IP:端口写入注册中心.
2. 当注册中心接收服务数据时,会动态的更新维护服务列表数据.
3. 当消费者启动服务时,会链接注册中心.获取服务列表数据.将数据保存到本地.为了下次调用时方便.
4. 当用户发起某项业务请求时,服务消费者根据服务的类型查找服务列表信息查看哪些服务器可以提供服务.之后利用负载均衡访问其中的一台服务器.直接调用服务生产者.
5. 当后台的服务提供者宕机.注册中心内部有心跳检测机制,发现服务器宕机则会动态的维护自身的服务列表数据,之后全网广播,更新消费者的服务列表.这种做的优点是用户永远不会访问故障机.从而实现了故障的迁移(高可用)

### 反向代理和正向代理的区别

反向代理:

用户发起请求时被nginx服务监控.**由第三方服务代替用户发起请求最终获取资源的过程.**

用户不知道到底访问的是哪台服务器!!!!! nginx

正向代理:

通过第三方代理软件.用户访问服务器时已经知道了服务器IP和端口.由用户直接发起请求的过程.

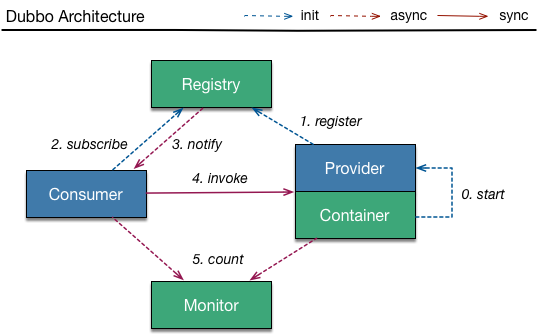
## Duubo框架介绍

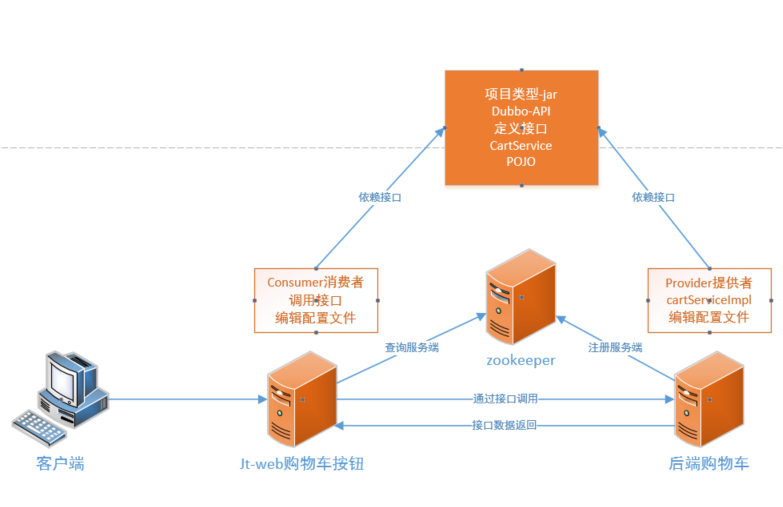
### 介绍

Dubbo是[1]  阿里巴巴公司开源的一个高性能优秀的[服务框架](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E6%A1%86%E6%9E%B6)(SOA)，使得应用可通过高性能的 **RPC** 实现服务的输出和输入功能，可以和 [2]  [Spring](https://baike.baidu.com/item/Spring)框架无缝集成。

Dubbo是一款高性能、轻量级的开源Java RPC框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

### Dubbo调用过程





## 注册中心安装使用

### Zookeeper介绍

ZooKeeper是一个[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F/19276232)的，开放源码的[分布式应用程序](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F/9854429)协调服务，是[Google](https://baike.baidu.com/item/Google)的Chubby一个[开源](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90/246339)的实现，是Hadoop和Hbase的重要组件。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。

ZooKeeper的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

ZooKeeper包含一个简单的原语集，[1]  提供Java和C的接口。

ZooKeeper代码版本中，提供了分布式独享锁、选举、队列的接口，代码在zookeeper-3.4.3\src\recipes。其中分布锁和队列有[Java](https://baike.baidu.com/item/Java/85979)和C两个版本，选举只有Java版本。（概述图片来源：[2]  ）

总结:Zookeeper是分布式的服务的**调度服务器**.

### Zookeeper介绍

ZooKeeper是一个[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F/19276232)的，开放源码的[分布式应用程序](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F/9854429)协调服务，是[Google](https://baike.baidu.com/item/Google)的Chubby一个[开源](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90/246339)的实现，是Hadoop和Hbase的重要组件。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。

ZooKeeper的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

ZooKeeper包含一个简单的原语集,提供Java和C的接口。

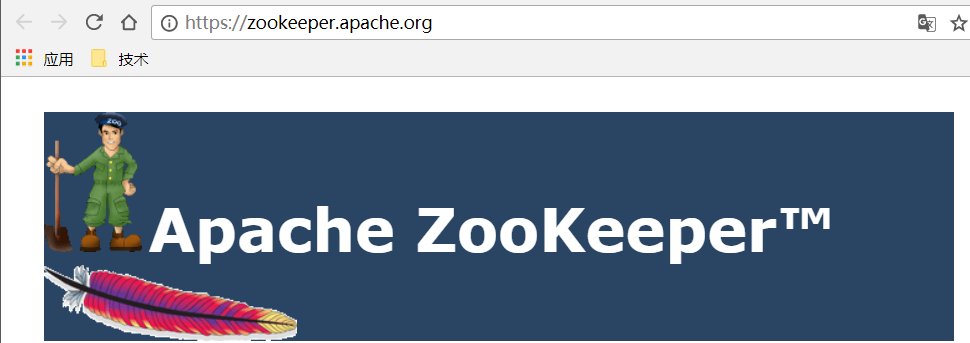
ZooKeeper代码版本中，提供了分布式独享锁、选举、队列的接口，代码在zookeeper-3.4.3\src\recipes。其中分布锁和队列有[Java](https://baike.baidu.com/item/Java/85979)和C两个版本，选举只有Java版本。

总结:Zookeeper负责服务的协调调度.当客户端发起请求时,返回正确的服务器地址.

### Zookeeper下载

网址: <http://zookeeper.apache.org/releases.html>.

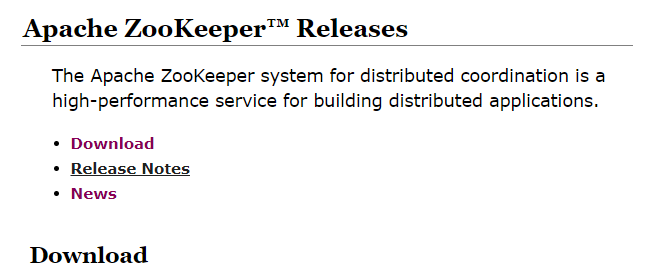
如图-2所示



**图-**2

下载路径,点击download.

如图-3所示

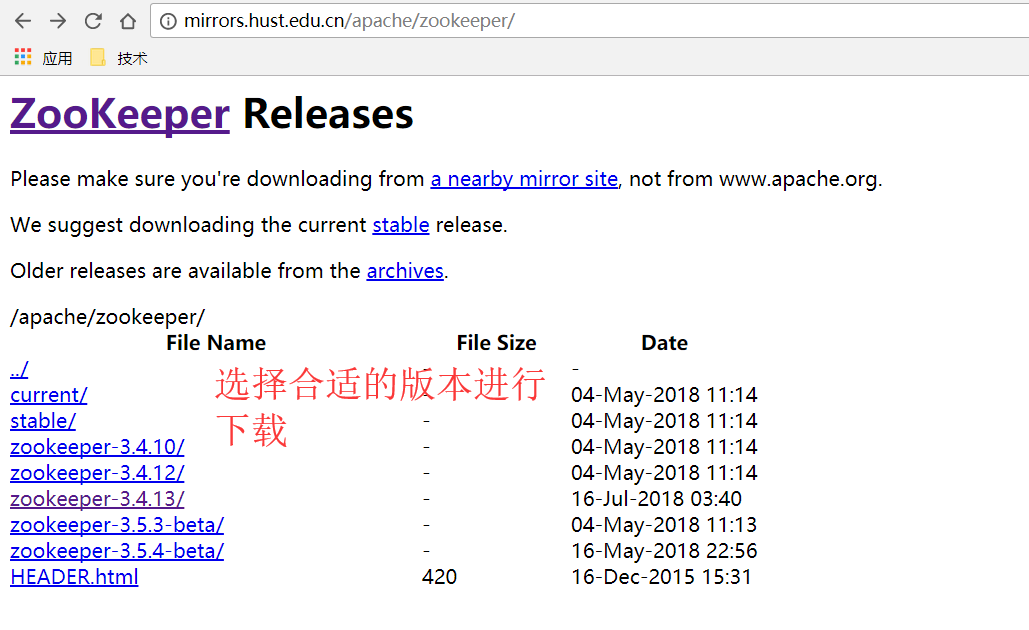


**图-**3

下载Zookeeper地址.

<http://mirrors.hust.edu.cn/apache/zookeeper/>

如图-4所示



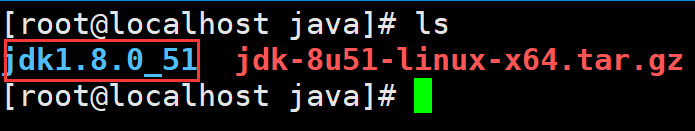
**图-**4

## Zookeeper安装

### 安装JDK

将JDK1.8文件上传到Linux操作系统中/src/usr/local/java/文件下.

如图-5所示



**图-**5

1.解压文件

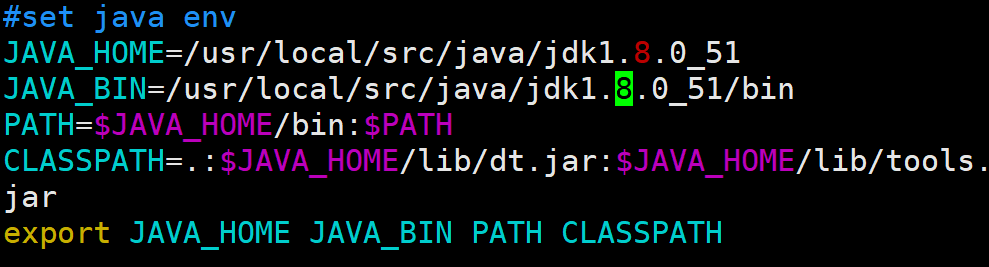
tar -xvf jdk-8u51-linux-x64.tar.gz

2.配置环境变量

编辑环境变量配置文件

vim /etc/profile

如图-6所示

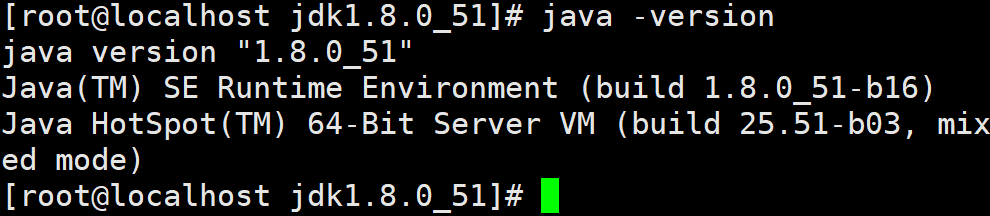


**图-**6

使JDK生效,之后检查jdk安装是否成功

source /etc/profile

如图 -7所示

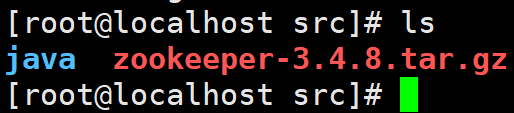


**图-**7

### 上传安装文件

说明:上传zookeeper安装文件.之后解压.

如图-8所示



**图-**8

解压目录:

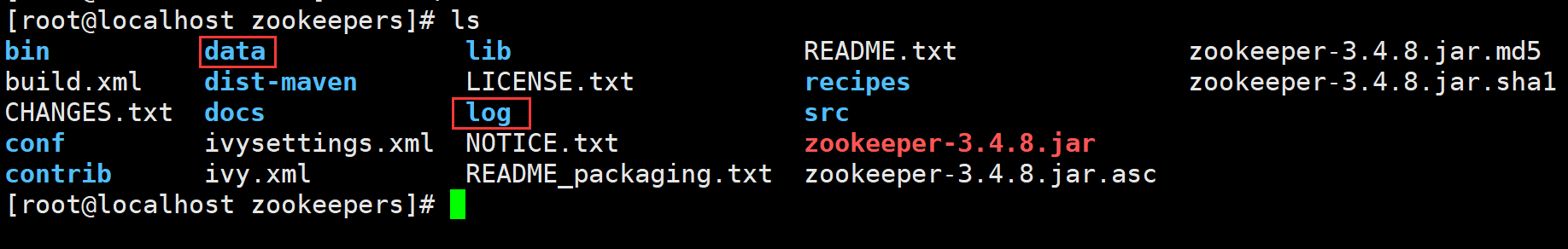
tar -xvf zookeeper-3.4.8.tar.gz

### 修改配置文件

在zk根目录下创建文件夹data/log

mkdir data log

如图-9所示

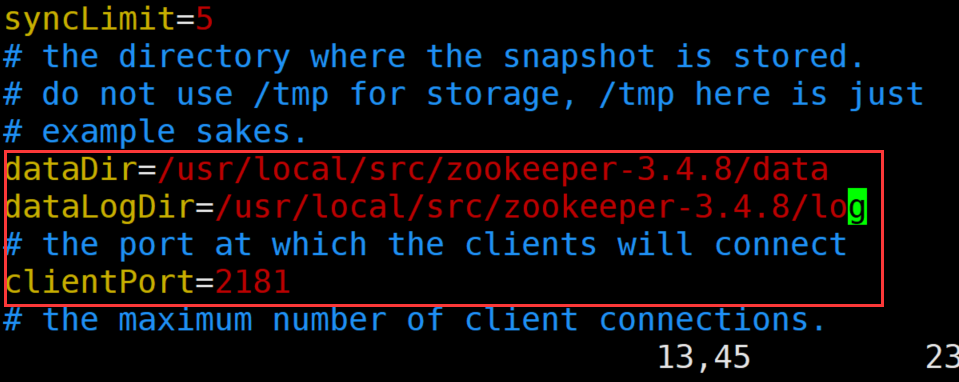


**图-**9

复制配置文件并且修改名称

cp zoo\_sample.cfg zoo.cfg

如图-10所示



**图-**10

### 启动zk

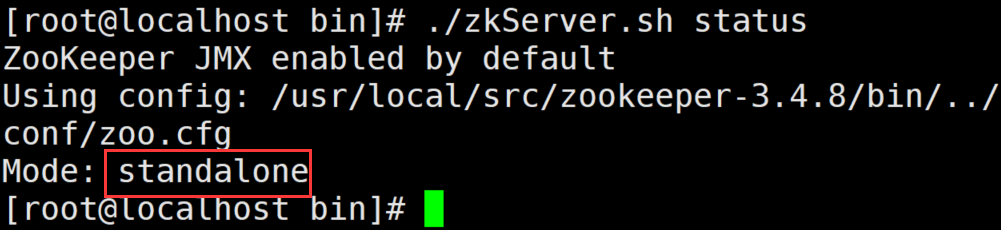
zk启动关闭命令如下.

sh zkServer.sh start 或者 ./zkServer.sh start

sh zkServer.sh stop

sh zkServer.sh status

如图-11所示



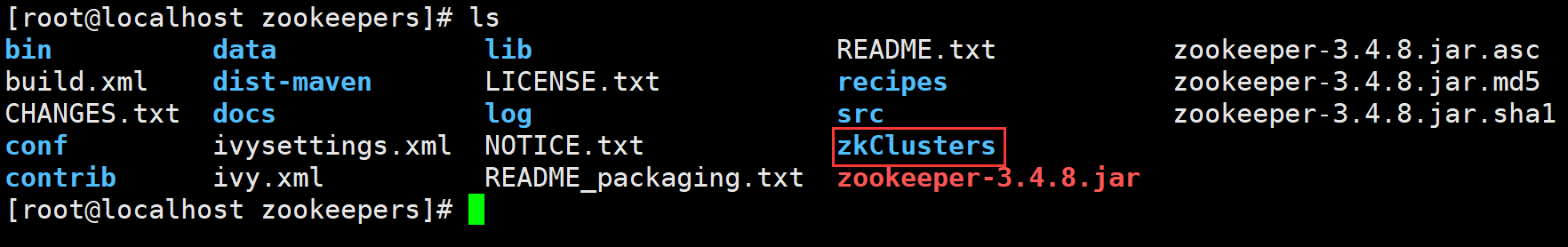
**图-**11

## Zookeeper集群安装

### 准备文件夹

在zookeeper根目录中创建新的文件夹zkCluster.

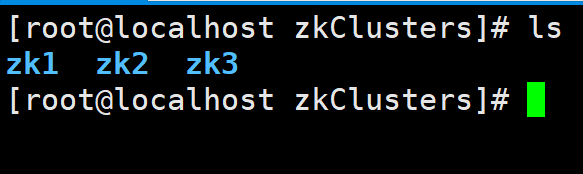
如图-12所示



**图-**12

如图-13所示

创建zk1/zk2/zk3文件夹.

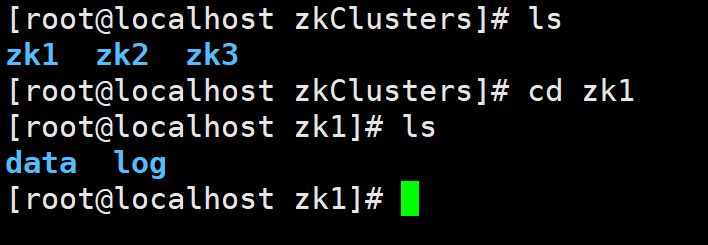


**图-**13

在每个文件夹里创建data/log文件夹.

mkdir {zk1,zk2,zk3}/{data,log}

如图-14所示

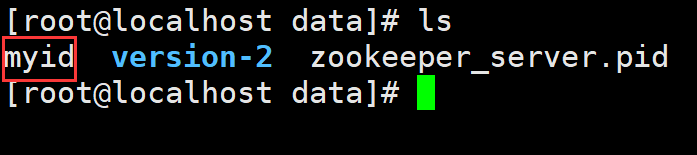


**图-**14

### 添加myid文件

分别在zk1/zk2/zk3中的data文件夹中创建新的文件myid.其中的内容依次为1/2/3,与zk节点号对应.

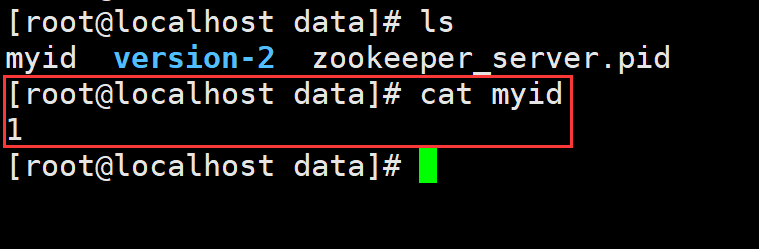
如图-15所示



**图-**15

编辑myid文件,定义编号.

如图-16所示

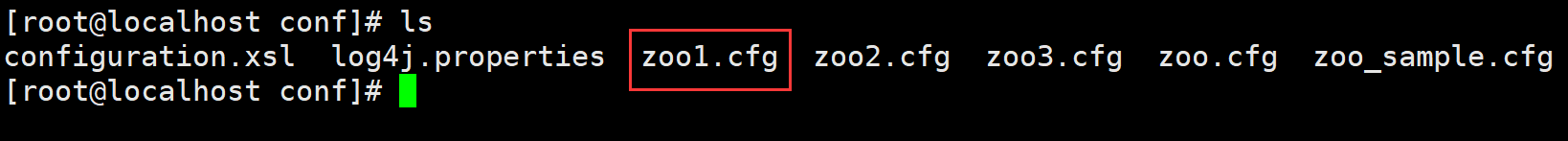


**图-**16

### 编辑配置文件

将zoo\_sample.cfg 复制为zoo1.cfg之后修改配置文件.

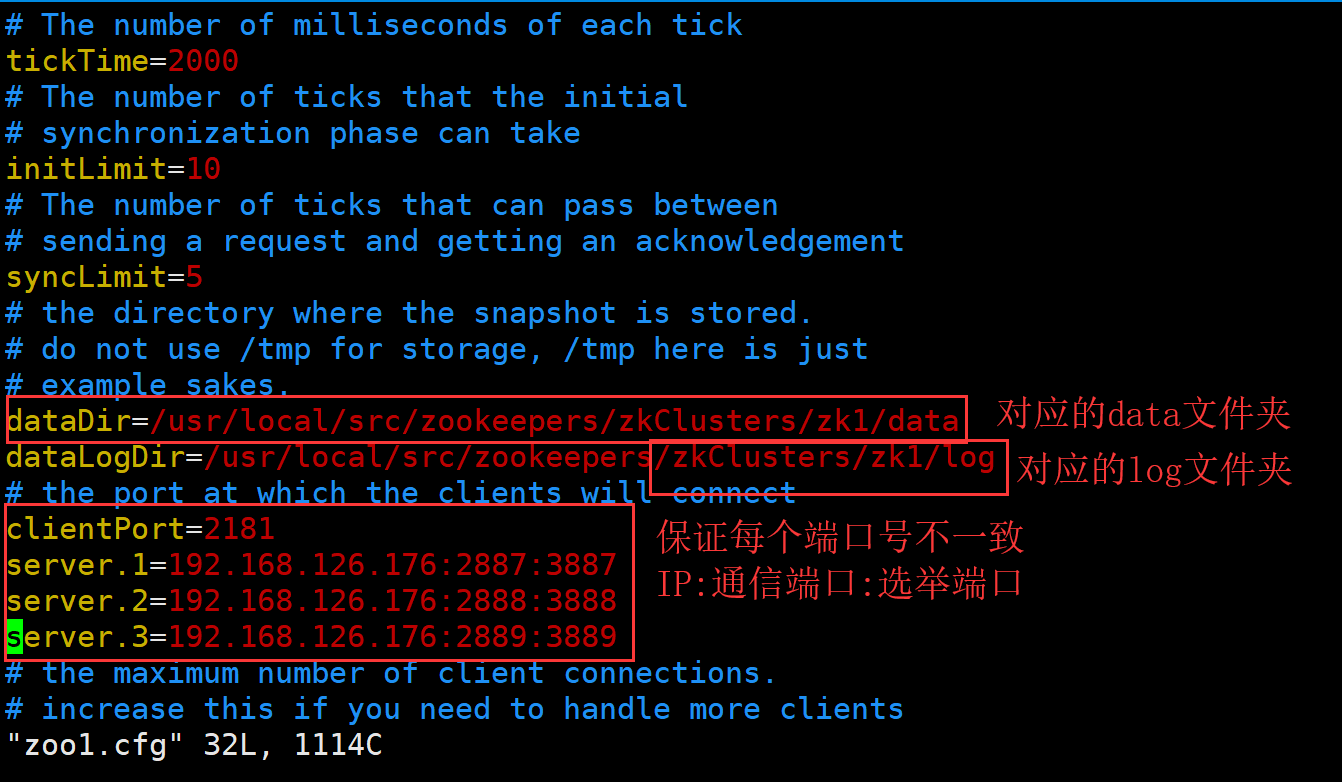
如图-17所示



**图-**17

### 修改zoo1.cfg

如图-18所示



**图-**18

配置完成后将zoo1.cfg复制2份.之后需要修改对应的文件夹目录.和不同的端口即可.

### ZK集群测试

通过下面的命令启动zk集群.

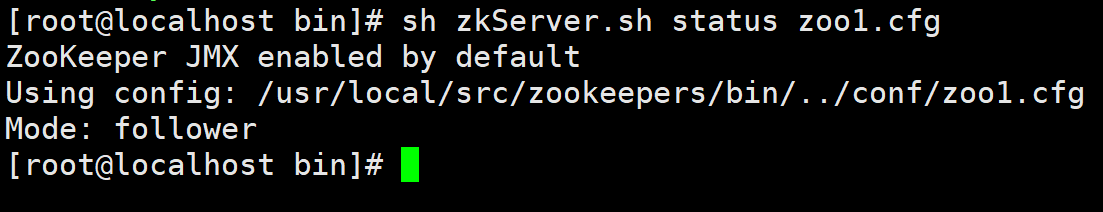
sh zkServer.sh start zoo1.cfg

sh zkServer.sh stop zoo1.cfg

sh zkServer.sh status zoo1.cfg

检查主从关系,从机情况说明.

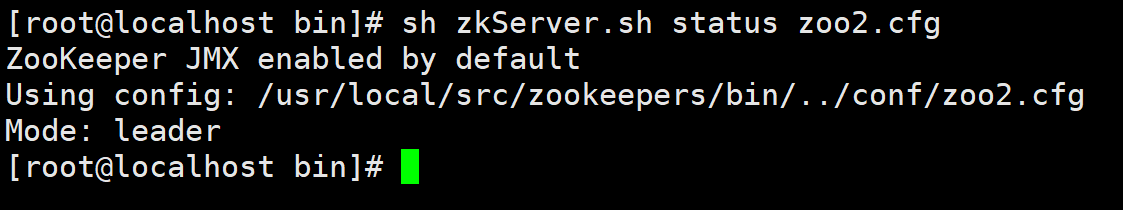
如图-19所示



**图-**19

检查主从关系,主机情况说明.

如图-20所示



**图-**20

### 关于zookeeper集群说明

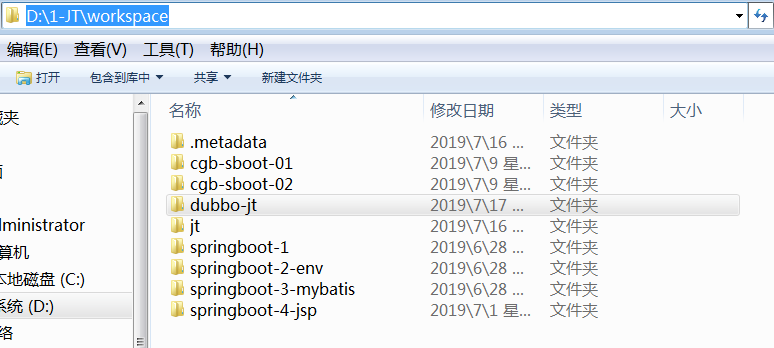
Zookeeper集群中leader负责监控集群状态,follower主要负责客户端链接获取服务列表信息.同时参与投票.

**图-**30

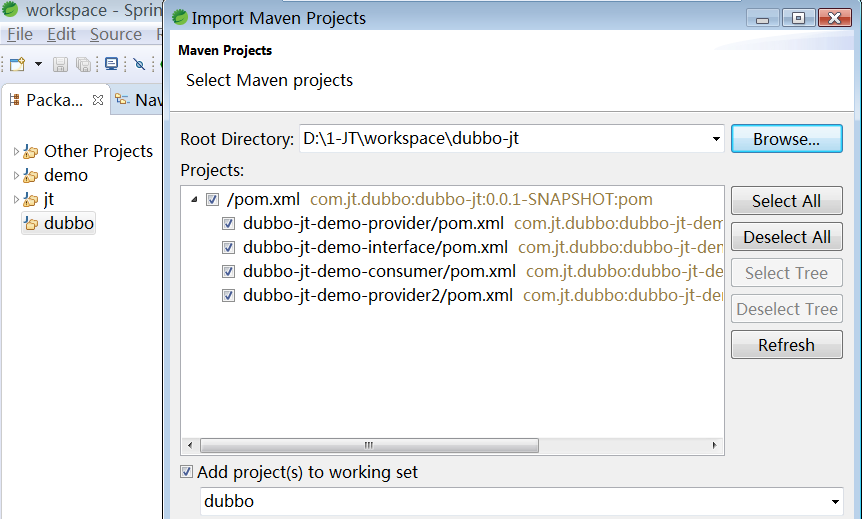
## Dubbo入门案例

### 导入入门案例

说明:将京淘课前资料中的dubbo-jt项目导入本地工作空间中.



2.导入项目



### 导入jar包

<!--引入dubbo配置 -->

<dependency>

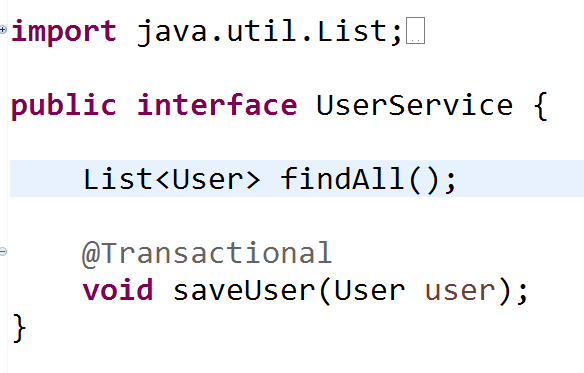
<groupId>com.alibaba.boot</groupId>

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

<version>0.2.0</version>

</dependency>

### 定义第三方接口



### 编辑服务生产者

说明:生产者与第三方接口是依赖,否则无法调用接口.

#### 编辑实现类

@Service(timeout=3000) //3秒超时

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** UserMapper userMapper;

@Override

**public** List<User> findAll() {

System.***out***.println("我是第一个服务的提供者");

**return** userMapper.selectList(**null**);

}

@Override

**public** **void** saveUser(User user) {

userMapper.insert(user);

}

}

#### 编辑生产者配置文件

server:

port: 9000

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

dubbo:

scan:

basePackages: com.jt

application:

name: provider-user #定义服务名称

registry: #定义注册中心链接

address: zookeeper://192.168.182.129:2181

protocol:

name: dubbo #变量名称一般固定

port: 20880 #生产者的端口号

mybatis-plus:

type-aliases-package: com.jt.dubbo.pojo #配置别名包路径

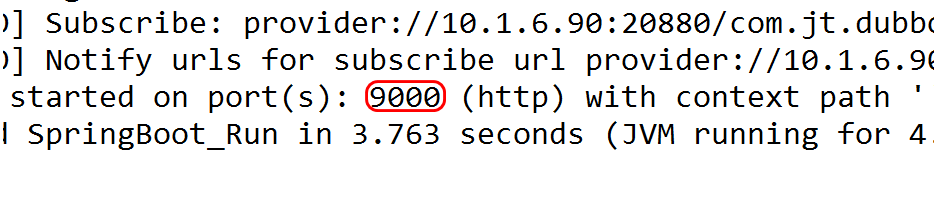
mapper-locations: classpath:/mybatis/mappers/\*.xml #添加mapper映射文件

configuration:

map-underscore-to-camel-case: true #开启驼峰映射规则

#### 启动测试

当启动主启动类,出现端口号表示服务一切正常.



#### 链接异常问题

如果报错出现链接异常.则检查防火墙是否关闭.

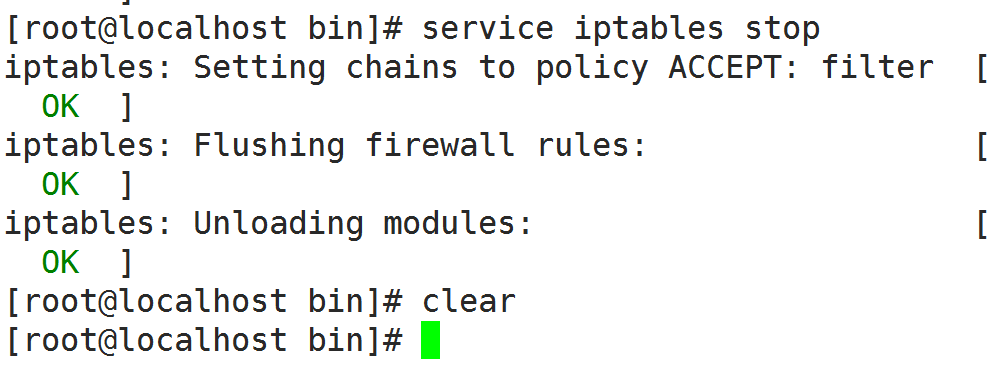
org.apache.curator.CuratorConnectionLossException: KeeperErrorCode = ConnectionLoss

at org.apache.curator.ConnectionState.checkTimeouts(ConnectionState.java:225) [curator-client-2.12.0.jar:na]

at org.apache.curator.ConnectionState.getZooKeeper(ConnectionState.java:94) [curator-client-2.12.0.jar:na]

at org.apache.curator.CuratorZookeeperClient.getZooKeeper(CuratorZookeeperClient.java:117) [curator-client-2.12.0.jar:na]

at org.apache.curator.framework.imps.CuratorFrameworkImpl.getZooKeeper(CuratorFrameworkImpl.java:489) [curator-framework-2.12.0.jar:na]



### 编辑服务消费者

#### 编辑Controller

@RestController

**public** **class** UserController {

/\*\*

\* timeout: 消费者调用服务提供者时 超时时间

\* check: true 当消费者调用时检查服务的提供者是否有效.

\*/

@Reference(timeout=3000,check=**true**)

**private** UserService userService;

@RequestMapping("/findAll")

**public** List<User> findAll(){

**return** userService.findAll();

}

@RequestMapping("/saveUser/{name}/{age}/{sex}")

**public** String saveUser(User user) {

userService.saveUser(user);

**return** "用户入库成功!!!";

}

}

#### 编辑YML配置文件

server:

port: 9001

dubbo:

scan:

basePackages: com.jt

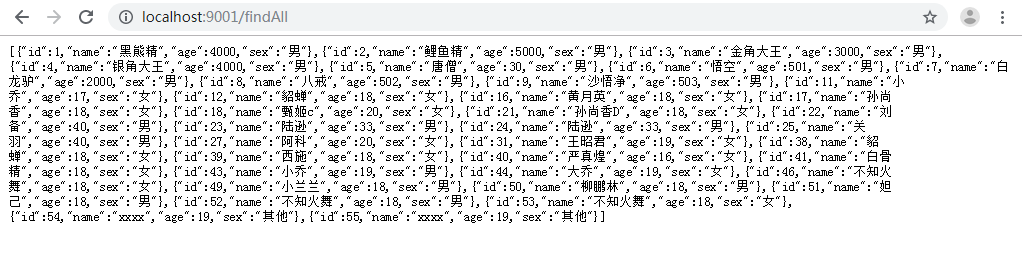
application:

name: consumer-user

registry:

address: zookeeper://192.168.182.129:2181

#### 远程调用测试



### Dubbo使用场景

说明:dubbo框架一般适用于自己的业务系统的调用.但是如果调用第三方接口(微信/支付宝/xxx)使用**HttpClient**

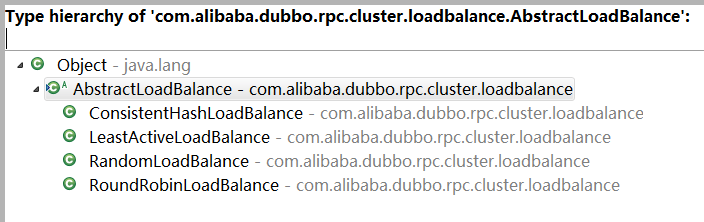
## dubbo负载均衡/高可用测试

### dubbo负载均衡策略说明

1. 默认条件下dubbo的负载均衡策略采用随机算法实现.

下列是dubbo的负载均衡策略.

**原则:编辑时使用类的前缀,并且首字母小写**



#### hash策略

@Reference(timeout=3000,check=**false**,loadbalance = "consistent")

**private** UserService userService;

#### 最少访问策略

@Reference(timeout=3000,check=**false**,loadbalance ="leastactive")

**private** UserService userService;

#### 轮询策略

@Reference(timeout=3000,check=**false**,loadbalance ="roundrobin")

**private** UserService userService;

#### 随机策略

@Reference(timeout=3000,check=**false**,loadbalance ="random")

**private** UserService userService;

### 服务高可用测试

当生产者宕机时,检查用户的访问是否受限.

答案:zookeeper已经实现了服务高可用.用户访问不受限制.

### 面试题

问:如果zookeeper宕机服务能否正常运行???

答案:服务依然可以正常运行!!!!

问题2:如果zk宕机之后生产者宕机,问用户是否受限???

答案:如果当前还有服务提供者,那么程序依然正常访问!!!

原因:

因为服务消费者启动时将注册中心中的服务列表数据缓存到本地.所以zk宕机时不受影响.

如果在zk宕机的同时生产者也宕机,则消费者在尝试调用生产者时发现服务器宕机.则在本地自动表示为down.之后的访问不会再次访问故障机.

但是现在程序运行处于高风险运行的状态.