【本次课程内容总结】

# SpringBoot上传文件路径映射

## 配置类,实现WebMvcConfigurer

|  |
| --- |
| @Configuration//配置类->配置文件  **public** **class** MyConfig **implements** WebMvcConfigurer{  @Override  **public** **void** addResourceHandlers(ResourceHandlerRegistry registry) {  //一个\*代表下一级 ,例如:/upload/\* 形如/upload/1.jpg  //2个\*代表下多级(含下一级) ,例如:/upload/\*\* 形如/upload/1.jpg /upload/2020/07/20200718/1.jpg  registry.addResourceHandler("/upload/\*\*").addResourceLocations("file:///D:/images/");  }  } |

## controller上传,上传到真实路径,数据库表存的是逻辑路径

|  |
| --- |
| **if** (photo!=**null** && photo.getSize()>0) {  File dir = **new** File("D:/images/");  **if** (!dir.exists()) {  dir.mkdirs();  }  File destFile = **new** File(dir,photo.getOriginalFilename());  **try** {  photo.transferTo(destFile);  student.setPhotopath("upload/"+photo.getOriginalFilename());  } **catch** (IllegalStateException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } |
|  |

# Dubbo

# 1. 软件架构的演进过程

软件架构的发展经历了由单体架构、垂直架构、SOA架构到微服务架构的演进过程，下面我们分别了解一下这几个架构。

## 1.1 单体架构



架构说明：

​ 全部功能集中在一个项目内（All in one）。

架构优点：

​ 架构简单，前期开发成本低、开发周期短，适合小型项目。

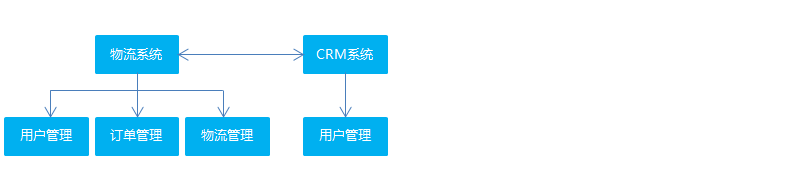
架构缺点：

​ 全部功能集成在一个工程中，对于大型项目不易开发、扩展和维护。

​ 技术栈受限，只能使用一种语言开发。

​ 系统性能扩展只能通过扩展集群节点，成本高。

## 1.2 垂直架构



架构说明：

​ 按照业务进行切割，形成小的单体项目。

架构优点：

​ 技术栈可扩展（不同的系统可以用不同的编程语言编写）。

架构缺点：

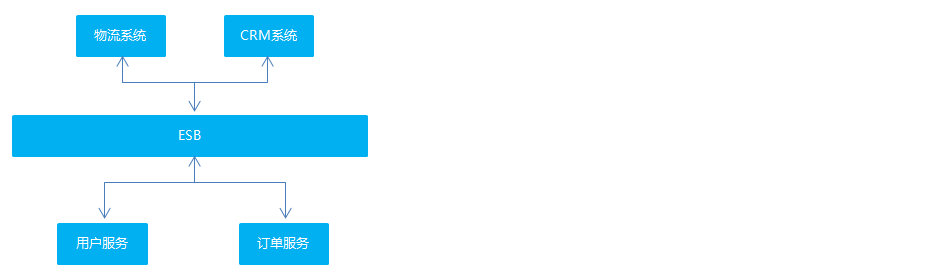
​ ​系统扩张只能通过集群的方式。

​ 项目之间功能冗余、数据冗余、耦合性强。

## 1.3 SOA架构

SOA全称为Service-Oriented Architecture，即面向服务的架构。它可以根据需求通过网络对松散耦合的粗粒度应用组件(服务)进行分布式部署、组合和使用。一个服务通常以独立的形式存在于操作系统进程中。

站在功能的角度，把业务逻辑抽象成可复用的服务，通过服务的编排实现业务的快速再生，目的：把原先固有的业务功能转变为通用的业务服务，实现业务逻辑的快速复用。



架构说明：

​ 将重复功能或模块抽取成组件的形式，对外提供服务，在项目与服务之间使用ESB（企业服务总线）的形式作为通信的桥梁。

架构优点：

​ 重复功能或模块抽取为服务，提高开发效率。

​ 可重用性高。

​ 可维护性高。

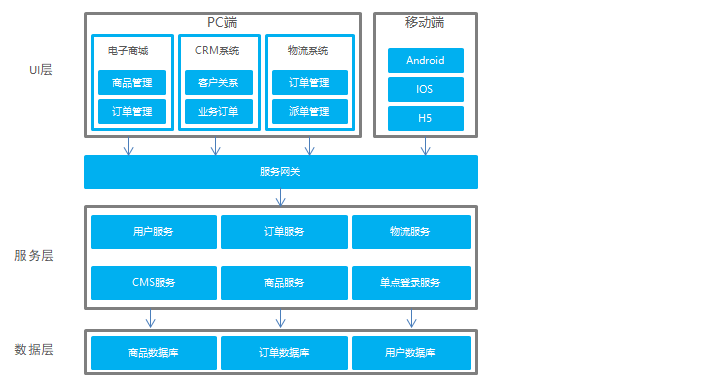
架构缺点：

​ 各系统之间业务不同，很难确认功能或模块是重复的。

​ 抽取服务的粒度大。

​ 系统和服务之间耦合度高。

## 1.4 微服务架构



架构说明：

​ 将系统服务层完全独立出来，抽取为一个一个的微服务。

​ 抽取的粒度更细，遵循单一原则。

​ 采用轻量级框架协议传输。

架构优点：

​ 服务拆分粒度更细，有利于提高开发效率。

​ 可以针对不同服务制定对应的优化方案。

​ 适用于互联网时代，产品迭代周期更短。

架构缺点：

​ 粒度太细导致服务太多，维护成本高。

​ 分布式系统开发的技术成本高，对团队的挑战大。

# 2. Apache Dubbo概述

## 2.1 Dubbo简介

Apache Dubbo是一款高性能的Java RPC框架。其前身是阿里巴巴公司开源的一个高性能、轻量级的开源Java RPC框架，可以和Spring框架无缝集成。

什么是RPC？

RPC全称为remote procedure call，即\*\*远程过程调用\*\*。比如两台服务器A和B，A服务器上部署一个应用，B服务器上部署一个应用，A服务器上的应用想调用B服务器上的应用提供的方法，由于两个应用不在一个内存空间，不能直接调用，所以需要通过网络来表达调用的语义和传达调用的数据。

需要注意的是RPC并不是一个具体的技术，而是指整个网络远程调用过程。

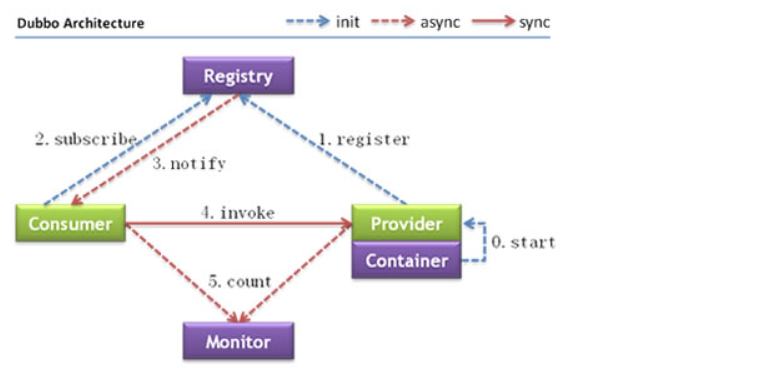
RPC是一个泛化的概念，严格来说一切远程过程调用手段都属于RPC范畴。各种开发语言都有自己的RPC框架。Java中的RPC框架比较多，广泛使用的有RMI、Hessian、Dubbo等。

Dubbo官网地址：http://dubbo.apache.org

Dubbo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。

## 2.2 Dubbo架构

Dubbo架构图（Dubbo官方提供）如下：



节点角色说明：

| 节点 | 角色名称 |

| --------- | ------------------- |

| Provider | 暴露服务的服务提供方 |

| Consumer | 调用远程服务的服务消费方 |

| Registry | 服务注册与发现的注册中心 |

| Monitor | 统计服务的调用次数和调用时间的监控中心 |

| Container | 服务运行容器 |

虚线都是异步访问，实线都是同步访问

蓝色虚线:在启动时完成的功能

红色虚线(实线)都是程序运行过程中执行的功能

调用关系说明:

0. 服务容器负责启动，加载，运行服务提供者。

1. 服务提供者在启动时，向注册中心注册自己提供的服务。

2. 服务消费者在启动时，向注册中心订阅自己所需的服务。

3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

# 3. 服务注册中心Zookeeper

通过前面的Dubbo架构图可以看到，Registry（服务注册中心）在其中起着至关重要的作用。Dubbo官方推荐使用Zookeeper作为服务注册中心。

## 3.1 Zookeeper介绍

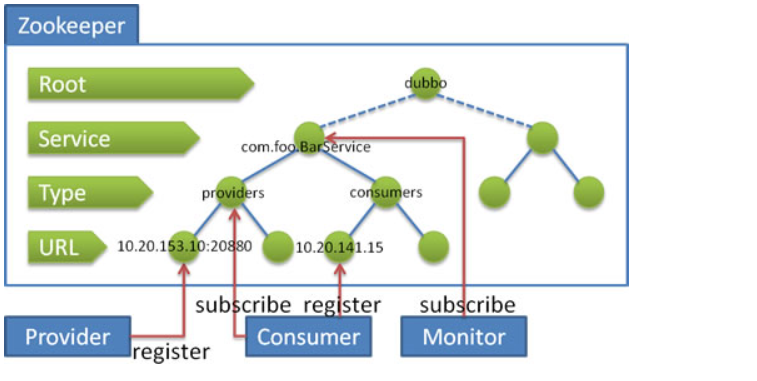
Zookeeper 是 Apache Hadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为 Dubbo 服务的注册中心，工业强度较高，可用于生产环境，并推荐使用 。

为了便于理解Zookeeper的树型目录服务，我们先来看一下我们电脑的文件系统(也是一个树型目录结构)：



我的电脑可以分为多个盘符（例如C、D、E等），每个盘符下可以创建多个目录，每个目录下面可以创建文件，也可以创建子目录，最终构成了一个树型结构。通过这种树型结构的目录，我们可以将文件分门别类的进行存放，方便我们后期查找。而且磁盘上的每个文件都有一个唯一的访问路径，例如：C:\Windows\aaa \bbb.txt。

Zookeeper树型目录服务：



流程说明：

- 服务提供者(Provider)启动时: 向 `/dubbo/com.foo.BarService/providers` 目录下写入自己的 URL 地址

- 服务消费者(Consumer)启动时: 订阅 `/dubbo/com.foo.BarService/providers` 目录下的提供者 URL 地址。并向 `/dubbo/com.foo.BarService/consumers` 目录下写入自己的 URL 地址

- 监控中心(Monitor)启动时: 订阅 `/dubbo/com.foo.BarService` 目录下的所有提供者和消费者 URL 地址

### 注册中心

• Multicast注册中心

• Zookeeper注册中心

• Redis注册中心

• Simple注册中心

### Dubbo优缺点

#### 优点：

• 透明化的远程方法调用

像调用本地方法一样调用远程方法；只需简单配置，没有任何API侵入

• 软负载均衡及容错机制

可在内网替代nginx lvs等硬件负载均衡器

• 服务注册中心自动注册 & 配置管理

不需要写死服务提供者地址，注册中心基于接口名自动查询提供者ip

• 使用类似zookeeper等分布式协调服务作为服务注册中心，可以将绝大部分项目配置移入zookeeper集群

• 服务接口监控与治理

Dubbo-admin与Dubbo-monitor提供了完善的服务接口管理与监控功能，针对不同应用的不同接口，可以进行 多版本，多协议，多注册中心管理

#### 缺点：

• 只支持JAVA语言

### Dubbo开发入门Demo

#### Zookeeper

##### 什么是Zookeeper

Zookeeper是一个分布式的服务框架,是树型的目录服务的数据存储，能做到集群管理数据 ，这里能很好的作为Dubbo服务的注册中心.

##### 使用Zookeeper作为Dubbo的注册中心的好处

• Dubbo能与Zookeeper做到集群部署，当提供者出现断电等异常停机时，Zookeeper注册中心能自动删除提供者信息，当提供者重启时，能自动恢复注册数据，以及订阅请求

3.2 安装Zookeeper

下载地址：http://archive.apache.org/dist/zookeeper/

本课程使用的Zookeeper版本为3.4.6，下载完成后可以获得名称为zookeeper-3.4.6.tar.gz的压缩文件。

安装步骤：

第一步：安装 jdk（略）

第二步：把 zookeeper 的压缩包（zookeeper-3.4.6.tar.gz）上传到 linux 系统

第三步：解压缩压缩包

​ tar -zxvf zookeeper-3.4.6.tar.gz

第四步：进入zookeeper-3.4.6目录，创建data目录

​ mkdir data

第五步：进入conf目录 ，把zoo\_sample.cfg 改名为zoo.cfg

​ cd conf

​ mv zoo\_sample.cfg zoo.cfg

第六步：打开zoo.cfg文件, 修改data属性：dataDir=/usr/local/zookeeper-3.4.6/data

3.3 启动、停止Zookeeper

进入Zookeeper的bin目录，启动服务命令

./zkServer.sh start

停止服务命令

./zkServer.sh stop

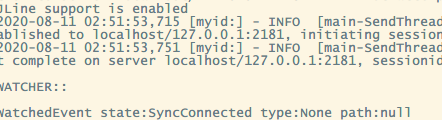
查看服务状态：

./zkServer.sh status

启动zookeeper的客户端

./zkCli.sh

客户端启动成功，就连上服务端了。



查看当前的所有节点，

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



继续查看dubbo下的子节点：

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

Windows下安装

1. 下载安装包
2. 解压到某个盘的某个目录下
3. 在解压后的文件夹新建2个文件夹:data和logs



1. 进入conf文件夹,将zoo\_sample.cfg复制一份并重命名为zoo.cfg
2. 打开zoo.cfg,改加:

|  |
| --- |
| dataDir=E:\\zookeeper-3.4.12\\data  dataLogDir=E:\\zookeeper-3.4.12\\logs |

1. 启动Zookeeper的服务:bin/zkServer.cmd

当出现:binding to port 0.0.0.0/0.0.0.0:2181代表Zookeeper服务启动成功

1. 启动Zookeeper客户端:bin/zkCli.cmd用命令启动Zookeeper客户端

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

[dubbo, zookeeper]

## 微服务架构

微服务是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成。系统中的各个微服务可被独立部署，各个微服务之间是松耦合的。每个微服务仅关注于完成一件任务并很好地完成该任务。在所有情况下，每个任务代表着一个小的业务能力。

例如要开发一个购票系统。就可以简单的拆分为用户管理微服务和售票系统微服务。两个服务都可以独立运行，都有自己的数据库，他们之间通过HTTP API 进行通信。

而SpringBoot就是搭建微服务的一个很好的选择。

4SpringBoot整合Dubbo,Zookeeper

4.1服务提供者工程

### 4.1.1创建Springboot工程

选择Spring Web场景启动器

### 4.1.2引入dubbo依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba.boot</groupId>  <artifactId>dubbo-spring-boot-starter</artifactId>  <version>0.2.0</version>  </dependency> |

### 4.1.3主类添加@EnableDubbo注解

|  |
| --- |
| @SpringBootApplication @EnableDubbo **public class** ProviderApplication {   **public static void** main(String[] args) {  SpringApplication.*run*(ProviderApplication.**class**, args);  } } |

开启Dubbo,让dubbo被Spring容器管理,能够在服务提供者工程发布服务.

### 4.1.4写接口

|  |
| --- |
| **package** com.icss.service;  **public interface** HelloService {   String sayHello(String name); } |

### 4.1.5写接口实现类

在实现类上添加@Component@Service,这样,只要启动应用,dubbo就会按照application.properties配置的地址发布到zookeeper上.

|  |
| --- |
| **package** com.icss.service.impl;  **import** com.alibaba.dubbo.config.annotation.Service; **import** com.icss.service.HelloService; **import** org.springframework.stereotype.Component;  @Component*//实例化到容器当中* @Service *//将服务注册到注册中心* **public class** HelloServiceImpl **implements** HelloService {   @Override  **public** String sayHello(String name) {  System.***out***.println(**"HelloServiceImpl.sayHello() ,"** + name);  **return "hello , "** + name;  } } |

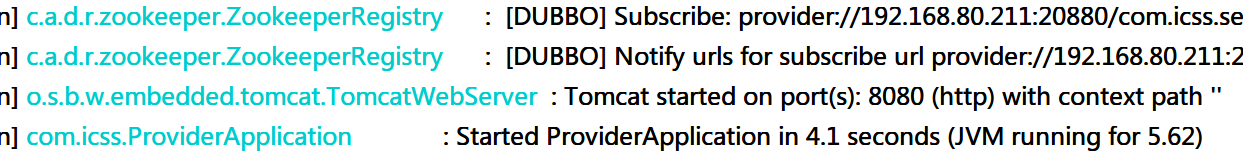
### 4.1.6将服务提供者注入到Dubbo注册中心

配置dubbo的扫描包和注册中心地址

|  |
| --- |
| *#将服务提供者注册到注册中心* **dubbo**:  *#配置服务提供者名称* **application**:  **name**: **"hello-provider"** *#注册中心* **registry**:  **address**: **"zookeeper://192.168.18.130:2181"** *#服务扫描的包* **scan**:  **base-packages**: **"com.icss.service"** |

### 4.1.7运行项目启动服务

1. 启动zookeeper
2. 运行主类



启动成功，注册成功

## 4.2服务消费者工程

### 4.2.1创建Springboot工程

选择Spring Web场景启动器

### 4.2.2导入dubbo依赖

### 4.2.3配置application.yml

|  |
| --- |
| *#服务器的端口号* **server**:  **port**: 8081 *#配置dubbo* **dubbo**:  **application**:  **name**: **"hello-consumer"  registry**:  **address**: **"zookeeper://192.168.18.130:2181"** |

### 4.2.4将服务端接口拷贝到客户端

|  |
| --- |
| **package** com.icss.service;  **public interface** HelloService {   String sayHello(String name); } |

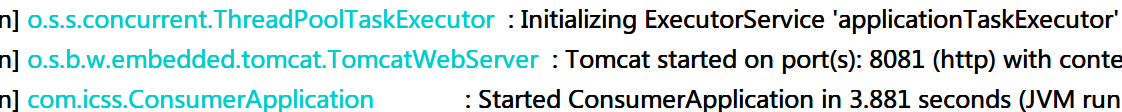
### 4.2.5编写Controller

使用@Reference引用接口调用服务.

|  |
| --- |
| **package** com.icss.controller;  **import** com.alibaba.dubbo.config.annotation.Reference; **import** com.icss.service.HelloService; **import** org.springframework.web.bind.annotation.PathVariable; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RestController;  @RestController **public class** HelloController {   *//注入service* @Reference*//从注册中心拉取服务* **private** HelloService **helloService**;   @RequestMapping(**"/hello/{name}"**)  **public** String hello(@PathVariable(**"name"**) String name){  **return helloService**.sayHello(name);  } } |

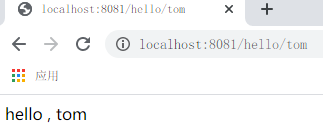
### 4.2.6启动

查看是否启动成功

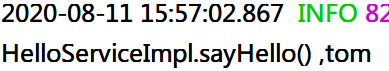


启动成功，订阅成功

### 4.2.7执行



服务提供者控制台输出：



# 5SpringBoot+Dubbo+Zookeeper分布式案例

案例:SpringBoot+Dubbo+Zookeeper

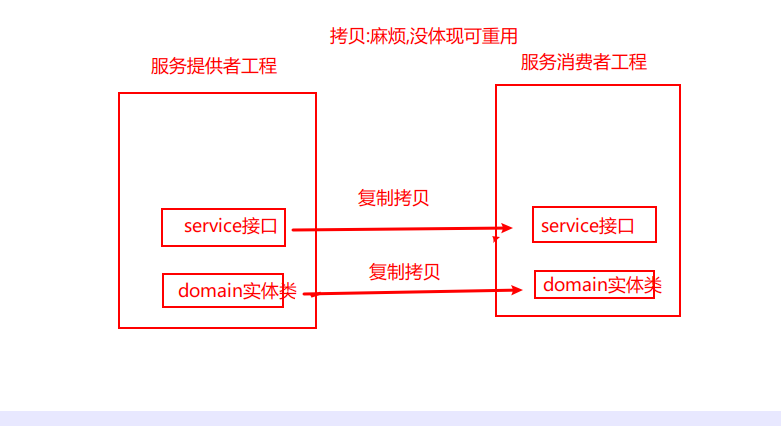
数据库表:user:id username password

服务提供者工程:list

服务消费者工程:查询服务端的列表数据

作法一：

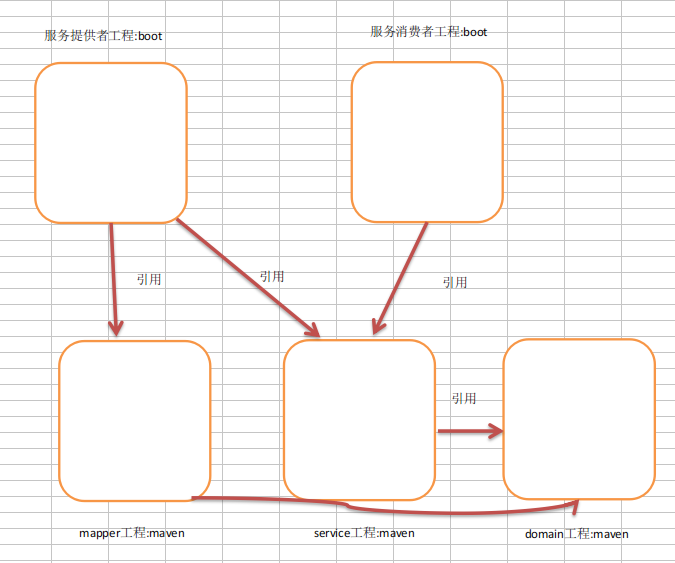
拷贝:笨拙,麻烦,没体现可重用理念



作法二：

引用:灵活,简单,体现了可重用的理念

1. 将包抽取出来放到一个独立的maven工程
2. 其他的工程如果想引用抽取出来的工程的话直接在pom.xml中加入抽取出来的工程依赖:dependency:gav坐标即可.



## 5.1数据准备

创建表user:uid username password

自动生成mapper和domain

创建5个工程:2个boot工程 3个maven工程

注意在这几个工程总，包名是统一的，不能一个工程一个包名。

## 5.2usermanagement-domain工程

该工程用来声明实体类，创建一个maven工程。

|  |
| --- |
| package com.icss.usermanagement.domain;  import java.io.Serializable;  public class User implements Serializable {   private Integer uid;  private String username;  private String password;   public User() {  }   public User(Integer uid, String username, String password) {  this.uid = uid;  this.username = username;  this.password = password;  }   public Integer getUid() {  return uid;  }   public void setUid(Integer uid) {  this.uid = uid;  }   public String getUsername() {  return username;  }   public void setUsername(String username) {  this.username = username;  }   public String getPassword() {  return password;  }   public void setPassword(String password) {  this.password = password;  }   @Override  public String toString() {  return "User{" +  "uid=" + uid +  ", username='" + username + '\'' +  ", password='" + password + '\'' +  '}';  } } |

## 5.3usermanagement-mapper工程

该工程用来处理数据访问，maven工程。

### 5.3.1导入依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.icss</groupId>  <artifactId>usermanagement-domain</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.47</version>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>2.1.3</version>  </dependency> </dependencies> |

### 5.3.2编写mapper接口

|  |
| --- |
| package com.icss.usermanagement.mapper;  import com.icss.usermanagement.domain.User; import org.apache.ibatis.annotations.Mapper; import org.apache.ibatis.annotations.Select;  import java.util.List;  @Mapper public interface UserMapper {   @Select("select \* from user")  List<User> findAll(); } |

## 5.4usermanagement-interface工程

该工程用来声明各种service的接口，maven工程。

### 5.4.1引入依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.icss</groupId>  <artifactId>usermanagement-domain</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency> </dependencies> |

### 5.4.2创建UserService接口

|  |
| --- |
| package com.icss.usermanagement.service;  import com.icss.usermanagement.domain.User;  import java.util.List;  public interface UserService {   */\*\*  \* 查询所有用户  \* @return  \*/* List<User> findAll(); } |

## 5.5服务提供者工程usermanagement-service

Springboot工程

### 5.5.1 导入依赖

|  |
| --- |
| <dependency>  <groupId>com.icss</groupId>  <artifactId>usermanagement-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>com.icss</groupId>  <artifactId>usermanagement-mapper</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba.boot</groupId>  <artifactId>dubbo-spring-boot-starter</artifactId>  <version>0.2.0</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  <exclusions>  <exclusion>  <groupId>org.junit.vintage</groupId>  <artifactId>junit-vintage-engine</artifactId>  </exclusion>  </exclusions>  </dependency> </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins> </build> |

### 5.5.2主类添加@EnableDubbo注解

|  |
| --- |
| @SpringBootApplication @EnableDubbo public class UsermanagementServiceApplication {   public static void main(String[] args) {  SpringApplication.*run*(UsermanagementServiceApplication.class, args);  }  } |

开启Dubbo,让dubbo被Spring容器管理,能够在服务提供者工程发布服务.

### 5.5.3编写service实现类

|  |
| --- |
| package com.icss.usermanagement.service.impl;  import com.alibaba.dubbo.config.annotation.Service; import com.icss.usermanagement.domain.User; import com.icss.usermanagement.mapper.UserMapper; import com.icss.usermanagement.service.UserService; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component; import org.springframework.transaction.annotation.Transactional;  import java.util.List;  @Component @Service public class UserServiceImpl implements UserService {   @Autowired  private UserMapper userMapper;   @Transactional(readOnly = true)  @Override  public List<User> findAll() {  return userMapper.findAll();  } } |

### 5.5.4编写application.yml配置文件

|  |
| --- |
| *#数据源配置* spring:  datasource:  driver-class-name: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://192.168.208.128:3306/dubbo?useUnicode=true&charactorEncoding=UTF8&serverTimezone=UTC  username: root  password: 123456 *#dubbo配置* dubbo:  *#应用名称* application:  name: usermanagement-service  *#注册中心地址* registry:  address: zookeeper://192.168.208.128:2181  *#服务扫描包* scan:  base-packages: com.icss.usermanagement.service |

## 5.6服务消费者工程usermanagement-controller

Springboot工程

### 5.6.1导入依赖

|  |
| --- |
| <dependencies> *<!-- 引入接口-->* <dependency>  <groupId>com.icss</groupId>  <artifactId>usermanagement-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba.boot</groupId>  <artifactId>dubbo-spring-boot-starter</artifactId>  <version>0.2.0</version>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  <exclusions>  <exclusion>  <groupId>org.junit.vintage</groupId>  <artifactId>junit-vintage-engine</artifactId>  </exclusion>  </exclusions>  </dependency>  </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

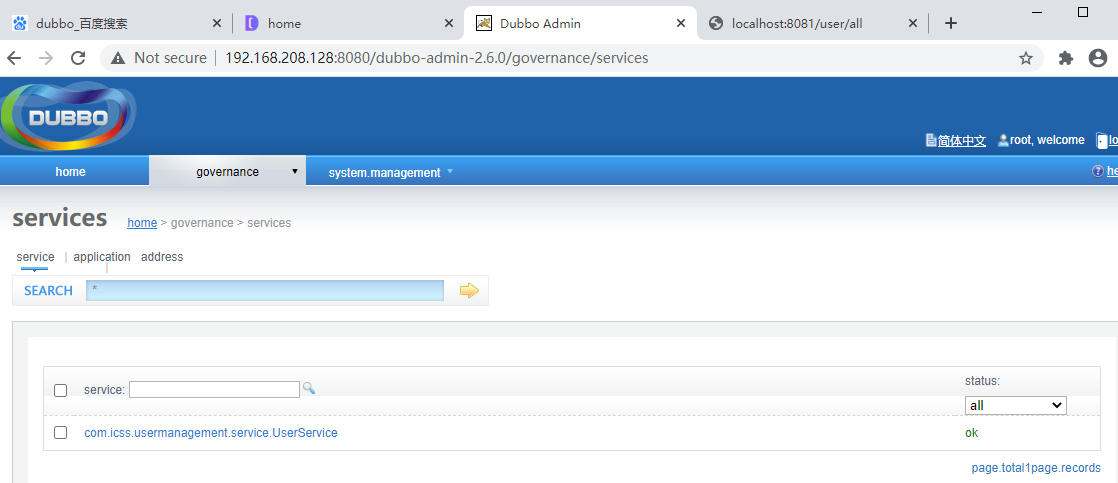
### 5.6.2创建controller

|  |
| --- |
| package com.icss.usermanagement.controller;  import com.alibaba.dubbo.config.annotation.Reference; import com.icss.usermanagement.domain.User; import com.icss.usermanagement.service.UserService; import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.ResponseBody;  import java.util.List;  @Controller @RequestMapping("/user") public class UserController {   @Reference*//从注册中心拉取服务* private UserService userService;   @RequestMapping("/all")  @ResponseBody  public List<User> findAll(){  return userService.findAll();  } } |

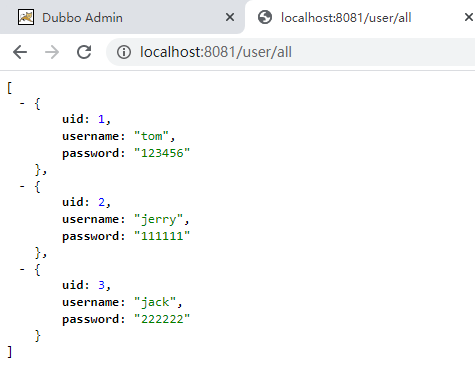
## 5.7测试执行

### 5.7.1启动服务提供者

### 5.7.2启动服务消费者



### 5.7.3访问controller



# 总结

1.SpringBoot文件上传到指定的目录

1.配置类

@Configuration

public class MyConfig implements WebMvcConfigurer{

public void addResourceHandler(ResourceHandlerRegistry registry){

registry.addResourceHandler("/upload/\*\*").addResourceLocations("file:///D:/images/");

}

}

2.Controller

File dir=new File("D:/images/");//真实路径

if(!dir.exists(){

dir.mkdirs();

}

String filename=photo.getOriginalFilename();

File destFile=new File(dir,filename);

photo.transferTo(destFile);

student.setPhotoPath("upload/"+filename);//逻辑路径

2.互联网业务模型

1.单一应用

2.垂直应用

3.分布式应用

4.流动计算

SOA:面向服务的架构 Service Orinted Architecture

3.什么是Dubbo

1.一款分布式服务框架

2.高性能和透明化的RPC远程服务调用方案

3.SOA服务治理方案

4.Dubbo的原理(工作流程,架构)

• 0.服务容器负责启动，加载，运行服务提供者。

• 1.服务提供者在启动时，向注册中心注册自己提供的服务。

• 2.服务消费者在启动时，向注册中心订阅自己所需的服务。

• 3.注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

• 4.服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

• 5.服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心

5.注册中心

• Multicast注册中心

• Zookeeper注册中心

• Redis注册中心

• Simple注册中心

6.Dubbo优缺点

优点：

• 透明化的远程方法调用

像调用本地方法一样调用远程方法；只需简单配置，没有任何API侵入

• 软负载均衡及容错机制

可在内网替代nginx lvs等硬件负载均衡器

• 服务注册中心自动注册 & 配置管理

不需要写死服务提供者地址，注册中心基于接口名自动查询提供者ip

• 使用类似zookeeper等分布式协调服务作为服务注册中心，可以将绝大部分项目配置移入zookeeper集群

• 服务接口监控与治理

Dubbo-admin与Dubbo-monitor提供了完善的服务接口管理与监控功能，针对不同应用的不同接口，可以进行 多版本，多协议，多注册中心管理

缺点：

• 只支持JAVA语言

7.什么是Zookeeper

Zookeeper是一个分布式的服务框架,是树型的目录服务的数据存储，能做到集群管理数据 ，这里能很好的作为Dubbo服务的注册中心.

8.使用Zookeeper作为Dubbo的注册中心的好处

Dubbo能与Zookeeper做到集群部署，当提供者出现断电等异常停机时，Zookeeper注册中心能自动删除提供者信息，当提供者重启时，能自动恢复注册数据，以及订阅请求

9.安装Zookeeper

1.下载安装包

2.解压到某个盘的某个目录下

3.在解压后的文件夹新建2个文件夹:data和logs

4.进入conf文件夹,将zoo\_sample.cfg复制一份并重命名为zoo.cfg

5.打开zoo.cfg,改加:

dataDir=E:\\zookeeper-3.4.12\\data

dataLogDir=E:\\zookeeper-3.4.12\\logs

6.启动Zookeeper的服务:bin/zkServer.cmd

当出现:binding to port 0.0.0.0/0.0.0.0:2181代表Zookeeper服务启动成功

7.启动Zookeeper客户端:bin/zkCli.cmd用命令启动Zookeeper客户端

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

[dubbo, zookeeper]

10.SpringBoot整合Dubbo,Zookeeper

1.搭建2个boot工程(提供者工程,消费者工程)+3个maven工程(重用)

2.服务提供者工程

1.pom.xml

1.mybatis

2.mysql

3.pagehelper

4.dubbo

5.zkclient

6.person-mapper

7.person-service

2.application.yml

数据源

dubbo:

providor:

application: 工程名

registry:

address: zookeeper://127.0.0.1:2181

scan:

base-packages: com.tjetc.service

3.主配置类:@EnableDubbo @MapperScan("com.tjetc.mapper")

4.接口实现类:@Component @Service

2.服务消费者工程

1.pom.xml

1.dubbo

2.zkclient

3.person-service

2.application.yml

dubbo:

application:

name: 工程名

registry:

address: zookeeper://127.0.0.1:2181

3.Controller

@Reference

private PersonService personService;