# 服务器部署Dubbo服务

前文《6Maven构建分布式工程》中，service-user已经通过手写的Main函数启动，但这也只是局限于开发环境，本文介绍如何将服务打成jar包，部署在Linux系统生产环境中，使用Dubbo自有的启动类对外提供服务。主要是对service-user的一些配置文件做修改，以达到目的。

## 1、pom.xml修改

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.leo.common</groupId>  <artifactId>common-parent</artifactId>  <version>1.0-SNAPSHOT</version>  <relativePath>../common-parent</relativePath>  </parent>  <description>用户服务的Service实现</description>  <groupId>org.leo.service</groupId>  <artifactId>service-user</artifactId>  <version>${service-user.version}</version>  <packaging>jar</packaging>  <name>service-user</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <log4j.leve>debug</log4j.leve>  <log4j.ale>debug</log4j.ale>  </properties>  <dependencies>  <dependency>  <groupId>org.leo.common</groupId>  <artifactId>common-base</artifactId>  <version>${common-base.version}</version>  </dependency>  <dependency>  <groupId>org.leo.common</groupId>  <artifactId>common-config</artifactId>  <version>${common-config.version}</version>  </dependency>  <dependency>  <groupId>org.leo.common</groupId>  <artifactId>common-core</artifactId>  <version>${common-core.version}</version>  </dependency>  <dependency>  <groupId>org.leo.common</groupId>  <artifactId>common-entity</artifactId>  <version>${common-entity.version}</version>  </dependency>  <dependency>  <groupId>org.leo.interface</groupId>  <artifactId>Interface-user</artifactId>  <version>${interface-user.version}</version>  </dependency>  <!-- 测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  <!-- 日志适配器 -->  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j-slf4j-impl</artifactId>  <scope>test</scope>  </dependency>  <!-- 日志实现 -->  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j-core</artifactId>  </dependency>  <!-- aspectj框架包 -->  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  </dependency>  <!-- Spring依赖 -->  <!-- 1)spring核心依赖 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <!--spring ioc依赖 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <!--2 spring坐标依赖 -->  <!--3 spring web相关依赖 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  </dependency>  <!--4spring dao层依赖 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  </dependency>  <!--5spring test相关依赖 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  </dependency>  <!-- mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  </dependency>  <!-- MySQL -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <!-- 数据库连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  </dependency>  <!-- dubbo 需要的jar start -->  <dependency>  <groupId>org.jboss.netty</groupId>  <artifactId>netty</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>dubbo</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  </dependency>  <dependency>  <groupId>com.101tec</groupId>  <artifactId>zkclient</artifactId>  </dependency>  <!-- dubbo 需要的jar end -->  </dependencies>  <build>  <finalName>service-user</finalName>  <resources>  <!-- 打包需要的配置文件 -->  <resource>  <targetPath>${project.build.directory}/classes</targetPath>  <directory>src/main/resources</directory>  <filtering>true</filtering>  <includes>  <include>\*\*/\*.xml</include>  <include>\*\*/\*.properties</include>  </includes>  </resource>  <!-- 结合com.alibaba.dubbo.container.Main，Main类会去classes/META-INF/spring找Spring的配置xml -->  <!-- 所以我们打包的时候需要把我们的 Spring配置xml复制到指定目录 -->  <resource>  <targetPath>${project.build.directory}/classes/META-INF/spring</targetPath>  <directory>src/main/resources/spring</directory>  <filtering>true</filtering>  <includes>  <include>applicationContext.xml</include>  </includes>  </resource>  </resources>  <pluginManagement>  <plugins>  <!-- 解决Maven插件在Eclipse内执行了一系列的生命周期引起冲突 -->  <plugin>  <groupId>org.eclipse.m2e</groupId>  <artifactId>lifecycle-mapping</artifactId>  <version>1.0.0</version>  <configuration>  <lifecycleMappingMetadata>  <pluginExecutions>  <pluginExecution>  <pluginExecutionFilter>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-dependency-plugin</artifactId>  <versionRange>[2.0,)</versionRange>  <goals>  <goal>copy-dependencies</goal>  </goals>  </pluginExecutionFilter>  <action>  <ignore />  </action>  </pluginExecution>  </pluginExecutions>  </lifecycleMappingMetadata>  </configuration>  </plugin>  </plugins>  </pluginManagement>  <plugins>  <!-- 打包jar文件时，配置manifest文件，加入lib包的jar依赖 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-jar-plugin</artifactId>  <configuration>  <classesDirectory>target/classes/</classesDirectory>  <archive>  <manifest>  <mainClass>com.alibaba.dubbo.container.Main</mainClass>  <!-- 打包时 MANIFEST.MF文件不记录的时间戳版本 -->  <useUniqueVersions>false</useUniqueVersions>  <addClasspath>true</addClasspath>  <classpathPrefix>lib/</classpathPrefix>  </manifest>  <manifestEntries>  <Class-Path>.</Class-Path>  </manifestEntries>  </archive>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-dependency-plugin</artifactId>  <executions>  <execution>  <id>copy-dependencies</id>  <phase>package</phase>  <goals>  <goal>copy-dependencies</goal>  </goals>  <configuration>  <type>jar</type>  <includeTypes>jar</includeTypes>  <useUniqueVersions>false</useUniqueVersions>  <outputDirectory>  ${project.build.directory}/lib  </outputDirectory>  </configuration>  </execution>  </executions>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <configuration>  <skip>true</skip>  </configuration>  </plugin>  </plugins>  </build>  </project> |

## 2、applicationContext.xml修改

原配置

|  |
| --- |
| ~~<import resource=~~*~~"applicationContext-dao.xml"~~* ~~/>~~  ~~<import resource=~~*~~" dubbo-provider.xml"~~* ~~/>~~ |

现改为

|  |
| --- |
| <import resource=*"classpath:spring/applicationContext-dao.xml"* />  <import resource=*"classpath:spring/dubbo-provider.xml"* /> |

原因是Dubbo的启动Main类会去classes/META-INF/spring加载Spring的配置XML，所以在打包的时候将src/main/resources里的applicationContext.xml复制到classes/META-INF/spring里。

原配置中引入同一文件夹下的两个配置xml，但是现在已经打包到classes/META-INF/spring里，所以需要在前面加上相对路径。

## 3、启动脚本

|  |
| --- |
| #!/bin/sh  ## java env  export JAVA\_HOME=/usr/local/java/jdk1.8.0\_131  export JRE\_HOME=$JAVA\_HOME/jre  ## service name  APP\_NAME=user  SERVICE\_DIR=/home/leo/project/service/$APP\_NAME  SERVICE\_NAME=service-$APP\_NAME  JAR\_NAME=$SERVICE\_NAME\.jar  PID=$SERVICE\_NAME\.pid  cd $SERVICE\_DIR  case "$1" in  start)  nohup $JRE\_HOME/bin/java -Xms256m -Xmx512m -jar $JAR\_NAME >/dev/null 2>&1 &  echo $! > $SERVICE\_DIR/$PID  echo "=== start $SERVICE\_NAME"  ;;  stop)  kill `cat $SERVICE\_DIR/$PID`  rm -rf $SERVICE\_DIR/$PID  echo "=== stop $SERVICE\_NAME"  sleep 5  ##  ## edu-service-aa.jar  ## edu-service-aa-bb.jar  P\_ID=`ps -ef | grep -w "$SERVICE\_NAME" | grep -v "grep" | awk '{print $2}'`  if [ "$P\_ID" == "" ]; then  echo "=== $SERVICE\_NAME process not exists or stop success"  else  echo "=== $SERVICE\_NAME process pid is:$P\_ID"  echo "=== begin kill $SERVICE\_NAME process, pid is:$P\_ID"  kill -9 $P\_ID  fi  ;;  restart)  $0 stop  sleep 2  $0 start  echo "=== restart $SERVICE\_NAME"  ;;  \*)  ## restart  $0 stop  sleep 2  $0 start  ;;  esac  exit 0 |

注意脚本在Linux系统中的权限。

## 4、Dubbo的负载均衡策略

1）随机，按照权重设置随机概率。调用量越大，分布越均匀。

2）轮询，会出现慢的提供者累积请求的问题。比如第一次请求分发给第一台快的提供者，处理完毕，第二次请求分发个第二台慢的提供者，没有处理完毕。第三次请求分发给第一台服务器，处理完毕，第四次请求分发给第二台，之前的请求依然没处理完毕，第四次请求积压。最好把慢的提供者权重调低。

3）最少活跃调用数，相同活跃数则随机。活跃数指调用前后的计数差。

使慢的提供者收到更少的请求，因为越慢的提供者的调用前后计数差越来越大。

比如：AB两个提供者都提维护一个计数器，各有10个请求发送给AB，这时AB的计数器都是10，每处理完一个请求，计数器都会减1，直至0。

而B处理的速度快，很快把10个请求处理完毕，计数器为0，而A处理慢，一个都还没处理完，计数器为10。

这时来了一个新的处理，自然而然地被分发到B。其实就是能者多劳策略。

4）一致性Hash，相同参数请求分发给同一个提供者。

如果提供者挂了，基于虚拟节点分发给其他提供者。

## 5、Dubbo的容错机制

1）failover 失败后，重试调取其他服务器的提供者，通过retries=”2”来配置重试次数。缺点是会带来更长的延迟。通常用于查询操作。

2）failfast 调用失败，立刻报错。常用于新增操作。

比如使用服务1新增一条记录完毕后，在返回的时候超时了，消费者认为失败，如果是failover，retries=”2”这种情况，就会向服务2再发送请求，重复新增。

可以将增删改等操作与查询操作拆分成两个服务，分别设置容错策略。

3）failsafe 失败安全，出现异常直接忽略，多用于写入审计日志等操作。

4）failback 失败自动恢复，后台记录失败的请求，定时重发。多用于消息通知操作，比如推送。

5）forking 并行调用多个服务，只要一个成功即返回。浪费更多的服务器资源。

6）broadcast 广播调用所有提供者，逐个调用，任意一台报错则报错。通常用于通知所有提供者更新缓存或日志等本地文件。

比如说在没有引用Redis等内存数据库的时候，为了增加查询速度，每个提供者都缓存了一些配置项、基础参数等，一旦有更新，就采用此方式通知所有提供者去重新加载。