第一章 微服务分布式系统架构之zookeeper与dubbo课程介绍

第一集 微服务分布式系统架构之zookeeper与dubbo课程介绍

- 简介：课程介绍和课程大纲讲解，讲课风格和重点内容理解技巧

第二集 技术选型和学后水平

- 课程所需基础和技术选型讲解，学完课程可以到达怎样的程度
- 技术选型

  1、IDEA JDK8 Maven SpringBoot基础 linux（centos7） dubbo virtualbox

  2、理解掌握并使用dubbo集成springboot进行开发，对于zookeeper有一定了解

  3、掌握学习的技巧和解决问题的思路

 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第二章 互联网架构演进和分布式系统基础知识

第一集 传统互联网架构到分布式架构的架构演变

- 简介：讲解单机应用和分布式应用架构演进基础知识
- 互联网时代演变过程

  pc时代-->移动互联网时代-->物联网时代
- 互联网架构演变过程
  - 传统单体应用

  缺点：系统耦合度高 开发效率随着时间的增长而降低 启动应用的时间长 依赖庞大 等等

  适用场景：初创公司、业务场景简单、功能单一、研发人员较少
  - 微服务

    优点：易开发、理解、维护独立部署和启动

    不足：

    需要管理多个服务、在架构的层面变得复杂

    服务切分之后、服务之间的调用可能需要去解决分布式事务的问题

第二集 微服务核心基础知识讲解

- 简介：讲解微服务核心知识 ：网关、服务发现注册、配置中心、链路追踪、负载均衡器、熔断
- 网关：路由转发 + 过滤器

  /api/v1/pruduct/ 商品服务 /api/v1/order/ 订单服务 /api/v1/user/ 用户服务
- 服务注册发现：调用和被调用方的信息维护
- 配置中心：管理配置，动态更新 application.properties
- 链路追踪：分析调用链路耗时

  例子：下单-》查询商品服务获取商品价格-》查询用户信息-》保存数据库
- 负载均衡器：分发负载
- 熔断：保护自己和被调用方

第三集 dubbo的坎坷历程

- 简介：介绍dubbo的相关历史，对dubbo形成初步认知
- 2011年由阿里开源dubbo首个版本 --dubbo-2.0.7
- 随后更新维护频繁
- 直到2014年发布 dubbo-2.4.11之后，在之后的几年里，几乎没见到更新
- 同期springboot、springcloud出现
- '消沉了很长一段时间后'，dubbo于2017年，凤凰磐涅，浴火重生,捐献给Apache

第四集 dubbo与spring-cloud比较

- 简介：对目前主流微服务框架进行对比，给技术选型提供相应的参考
- dubbo: zookeeper + dubbo + springmvc/springboot

  官方地址：http://dubbo.apache.org/#!/?lang=zh-cn 配套 通信方式：rpc 性能高 注册中心：zookeper/redis 配置中心：diamond（企业很少用）
- springcloud: 全家桶+轻松嵌入第三方组件(Netflix 奈飞)

  官网：http://projects.spring.io/spring-cloud/ 配套 通信方式：http restful 注册中心：eruka/consul 配置中心：config 断路器：hystrix 网关：zuul 分布式追踪系统：sleuth+zipkin

第三章 课程基础环境搭建

第一集 virtualbox的下载安装

- 介绍virtualbox的下载及安装
- 下载地址 https://download.virtualbox.org/virtualbox/5.2.22/VirtualBox-5.2.22-126460-Win.exe
- 安装过程详见视频内容

第二集 virtualbox下安装centos7

- 简介：介绍virtualbox下安装centos7虚拟机
- centos7下载地址：http://mirrors.umflint.edu/CentOS/7.5.1804/isos/x86_64/CentOS-7-x86_64-Minimal-1804.iso
- 安装过程详见视频内容

第三集 配置虚拟机网络

- 简介：介绍怎么在配置虚拟机网络，达到宿主机与虚拟机互通
- 主机网络管理

  记住ipv4掩码 192.1.1.1 为虚拟机分配IP的时候，就在192.1.1.1 这个上面分配ip地址
- 设置--》网络--》启动网卡2--》选择hostOnly
- 启动虚拟机

cd /etc/sysconfig/network-scripts/ vi ifcfg-enp0s3 ONBOOT=yes cp ifcfg-enp0s3 ifcfg-enp0s8 vi ifcfg-enp0s8 TYPE=Ethernet BOOTPROTO=static IPADDR=192.1.1.101 NETMASK=255.255.255.0 NAME=enp0s8 DEVICE=enp0s8 ONBOOT=yes service network restart

第四集 实战--linux上安装java环境

- 下载jdk8

  https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- 配置环境变量

tar -zxvf jdk-8u191-linux-x64.tar.gz mv jdk1.8.0_191/ /usr/local/ vim /etc/profile

在文件的最后面加上下面几行

```
JAVA_HOME=/usr/local/jdk1.8.0_191
JRE_HOME=$JAVA_HOME/jre
PATH=$JAVA_HOME/bin:$PATH
export PATH
```

- 激活配置

  source /etc/profile

第五集 克隆多台虚拟机

- 简介：介绍如何使用virtualbox的克隆功能，快速克隆多台虚拟机，从而带到模拟集群的效果
- 在已安装的虚拟机上配置hosts

  vim /etc/hosts 在最下面，增加 192.1.1.101 xdclass1 192.1.1.102 xdclass2 192.1.1.103 xdclass3 保存退出之后用shutdown -h now 关闭虚拟机
- 在virtualBox上，点击管理，导出虚拟电脑，选择相应目录进行导出
- 导出完成后，点击管理，导入，选择刚刚导出的文件
- 按步骤2导入2台虚拟机
- 修改各自的ip

  vim /etc/sysconfig/network-scripts/ifcfg-enp0s8

  修改ip之后 service network restart‘

第四章 分布式应用程序注册中心之zookeeper

第一集 什么是微服务的注册中心

- 简介：讲解什么是注册中心，常用的注册中心有哪些

  理解注册中心：服务管理,核心是有个服务注册表，心跳机制动态维护

  服务提供者provider: 启动的时候向注册中心上报自己的网络信息

  服务消费者consumer: 启动的时候向注册中心上报自己的网络信息，拉取provider的相关网络信息

- 为什么要用

  微服务应用和机器越来越多，调用方需要知道接口的网络地址，如果靠配置文件的方式去控制网络地址，对于动态新增机器，维护带来很大问题

- 主流的注册中心

  zookeeper、Eureka、consul、etcd 等

第二集 分布式应用知识CAP理论知识

- 简介：讲解分布式核心知识CAP理论，包括一致性的多种分类等核心知识
- 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），三者不可同时获得。

- 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。（所有节点在同一时间的数据完全一致，越多节点，数据同步越耗时）

  强一致性（strong consistency）。任何时刻，任何用户都能读取到最近一次成功更新的数据。 单调一致性（monotonic consistency）。任何时刻，任何用户一旦读到某个数据在某次更新后的值，那么就不会再读到比这个值更旧的值。也 就是说，可获取的数据顺序必是单调递增的。

  会话一致性（session consistency）。任何用户在某次会话中，一旦读到某个数据在某次更新后的值，那么在本次会话中就不会再读到比这值更旧的值会话一致性是在单调一致性的基础上进一步放松约束，只保证单个用户单个会话内的单调性，在不同用户或同一用户不同会话间则没有保障。

  最终一致性（eventual consistency）。用户只能读到某次更新后的值，但系统保证数据将最终达到完全一致的状态，只是所需时间不能保障。

  弱一致性（weak consistency）。用户无法在确定时间内读到最新更新的值。

- 可用性（A）：负载过大后，集群整体是否还能响应客户端的读写请求。（服务一直可用，而且是正常响应时间）
- 分区容错性（P）：分区容忍性，就是高可用性，一个节点崩了，并不影响其它的节点（100个节点，挂了几个，不影响服务，越多机器越好）
- CAP理论就是说在分布式存储系统中，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

第三集 win、linux双环境安装zookeeper

- 简介：介绍如何在windows和linux环境下面安装zookeeper，方便开发调试以及正式部署
- 下载zookeeper https://zookeeper.apache.org/
- 配置jdk 参考以下博客配置

  https://blog.csdn.net/qq_35246620/article/details/61208961

- 解压下载好的zk
- 重命名conf目录下的zoo_sample.cfg 文件为zoo.cfg 并修改里面的内容

- 直接运行bin目录下的zkServer，单一实例便安装好
- 主要配置项

  tickTime 心跳基本时间单位，毫秒级，ZK基本上所有的时间都是这个时间的整数倍。

  initLimit 集群中的follower服务器(F)与leader服务器(L)之间初始连接时能容忍的最多心跳数

  syncLimit 集群中的follower服务器与leader服务器之间请求和应答之间能容忍的最多心跳数

  dataDir 内存数据库快照存放地址，如果没有指定事务日志存放地址(dataLogDir)，默认也是存放在这个 路径下，建议两个地址分开存放到不同的设备上。
- centos下安装zk

  tar -zxvf zookeeper-3.4.12.tar.gz -C /usr/local/

  cd /usr/local/zookeeper-3.4.12/conf

  mv zoo_sample.cfg zoo.cfg

  vim zoo.cfg

  修改dataDir /usr/local/zookeeper-3.4.12/data

  将整个文件夹所属权赋予zookeeper用户 chown -R zookeeper:zookeeper /usr/local/zookeeper-3.4.12

  切换到zookeeper用户 su zookeeper

  cd /usr/local/zookeeper-3.4.12/bin 找到对应的zkServer.sh启动

第四集 必知必会之ZooKeeper数据模型

- 简介：介绍zookeeper的数据模型，对zookeeper节点的特性做详细的介绍
- zookeeper的数据模型类似于linux下的文件目录

  /usr/local /usr/local/tomcat
- 每一个节点都叫做zNode，可以有子节点，也可以有数据
- 每个节点都能设置相应的权限控制用户的访问
- 每个节点存储的数据不宜过大
- 每个节点都带有一个版本号，数据变更时，版本号变更（乐观锁）
- 节点分永久节点跟临时节点

第五、六集 zookeeper常用命令之zkCli

- 简介：介绍zkCli客户端非常常用的命令
- zkCli.sh

  不填后面的参数，默认连接的就是localhost:2181
- win下面运行的是.cmd结尾的文件，linux下运行的是.sh结尾的文件

- 连接远程服务器

  zkCli.sh -timeout 0 -r -server ip:port

- zkCli.sh h 出现相应的帮助信息

  ZooKeeper -server host:port cmd args stat path [watch] set path data [version] ls path [watch] delquota [-n|-b] path ls2 path [watch] setAcl path acl setquota -n|-b val path history redo cmdno printwatches on|off delete path [version] sync path listquota path rmr path get path [watch] create [-s] [-e] path data acl addauth scheme auth quit getAcl path close connect host:port

- zookeeper节点

  /usr/local

- 创建节点 create [-s] [-e] path data acl -s表示创建顺序节点 -e表示创建临时节点 data表示创建的节点的数据内容

- 查看

  获取节点的子节点 ls path

  获取节点的数据 get path

  查看节点状态 stat path

  cZxid = 0x3 --事务id ctime = Tue Dec 04 10:37:58 EST 2018 --节点创建的时候的时间 mZxid = 0x3 --最后一次更新时的事务id mtime = Tue Dec 04 10:37:58 EST 2018 最后一次更新的时间 pZxid = 0x3 该节点的子节点列表最后一次被修改的事务id cversion = 0 子节点列表的版本 dataVersion = 0 数据内容的版本 aclVersion = 0 acl版本 ephemeralOwner = 0x0 用于临时节点，表示创建该临时节点的事务id，如果当前的节点不是临时节点，该字段值为0 dataLength = 8 数据内容的长度 numChildren = 0 子节点的数量

- 获取节点的子节点以及当前节点的状态 ls2 path
- 修改节点的数据 set path data [version]
- 删除节点数据

  delete path [version] 删除节点，如果此时该节点有子节点，则不允许删除

  rmr path 递归删除整个节点

第七集 深入理解zookeeper session机制

- 简介：介绍zookeeper的session机制，对该机制形成一定的认知
- 用于客户端与服务端之间的连接，可设置超时时间,通过心跳包的机制（客户端向服务端ping包请求）检查心跳结束，session就过期
- session过期的时候，该session创建的所有临时节点都会被抛弃

第八集 深入理解zookeeper watcher机制

- 简介：介绍zookeeper的watcher机制，以及如何使用
- 对节点的watcher操作 get stat

针对每一个节点的操作，都可以有一个监控者，当节点发生变化，会触发watcher事件 zk中watcher是一次性的，触发后立即销毁 所有有监控者的节点的变更操作都能触发watcher事件

- 子节点的watcher操作（监控父节点，当父节点对应的子节点发生变更的时候，父节点上的watcher事件会被触发） ls ls2 增删会触发、修改不会，如果子节点再去新增子节点，不会触发（也就是说，触发watcher事件一定是直系子节点）

第九、十集 zookeeper的acl（access control lists）权限控制

- 针对节点可以设置相关的读写等权限，目的是为了保证数据的安全性
- 权限permissions可以指定不同的权限范围及角色
- 常用的命令

  getAcl 获取节点权限 setAcl 设置节点权限 addauth 输入认证授权信息，注册时输入明文密码，但是在zk里，以加密的形式保存
- acl的组成 [scheme ID permissions]
- scheme 授权机制

  world下只有一个id，也就是anyone，表示所有人 world:anyone:permissions

  auth 代表认证登录，需要注册用户有权限才可以 auth:user:password:permissions

  digest 需要密码加密才能访问 digest:username:BASE64(SHA1(password)):permissions(跟auth区别在于，auth明文，digest为密文)

  ip ip:localhost:psermissions

  super 代表超管，拥有所有的权限；

  打开zk目录下的/bin/zkServer.sh服务器脚本文件，找到如下一行：

  ```
  nohup $JAVA "-Dzookeeper.log.dir=${ZOO_LOG_DIR}" "-
  Dzookeeper.root.logger=${ZOO_LOG4J_PROP}"
  加上 "-
  Dzookeeper.DigestAuthenticationProvider.superDigest=super:xQJmxLMiHGwaqBvst5y6rkB6HQs="
  此处后面的密文为密码加密后的
  ```
- id：允许访问的用户
- permissions：权限组合字符串

  ```
  cdrwa
  c create 创建子节点
  d delete 删除子节点
  r read 读取节点的数据
  w write 写入数据
  a admin 可管理的权限
  cdrwa cdr cdw
  ```
- acl的使用场景

  开发环境跟测试环境，使用acl就可以进行分离，开发者无权去操作测试的节点

  生产环境上控制指定ip的服务可以访问相关的节点

第五章 微服务核心知识之zookeeper选举机制

第一集 zk集群核心知识之三种角色及其作用

- 简介：讲解zookeeper集群中的三种角色以及其各自的作用
- leader：作为整个zk集群写请求的唯一处理者，并负责进行投票的发起和决议，更新系统的状态。
- follower：接收客户端请求，处理读请求，并向客户端返回结果；将写请求转给 Leader；在选举 Leader过程中参与投票。
- observer：可以理解为无选举投票权的 Flollower，其主要是为了协助 Follower 处理更多的读请求。如果 Zookeeper 集群的读请求负载很高，或者客户端非常非常多，多到跨机房，则可以设置一些 Observer 服务器，以提高读取的吞吐量。

第二集 zk核心知识之注册中心常见的三种模式

- 简介：介绍在zookeeper集群中常见的三种模式及各种模式出现的时机
- k的核心是广播机制，该机制保证了各个zk之间数据同步（数据一致性）。zk实现的机制为ZAB协议
- 恢复模式： 如果leader崩溃，这个时候就会进入恢复模式，使整个zk集群恢复到正常的工作状态
- 同步模式：新的leader选举出来后，就乎进入同步模式（各个follower会去同步新的leader上的数据），当大多数zkServer完成了与leader的状态同步之后，恢复模式就结束
- 广播模式：客户端想写入数据，这个时候leader发起提议，当leader的提议被大多数的zkServer统一之后，leader就会去修改自身的数据，并将修改后的数据广播给其他的follower

第三集 zk集群选举核心概念及选举时状态

- 简介：介绍zookeeper集群选举相关的核心概念以及选举时的状态
- myid

  这是 zk 集群中服务器的唯一标识，称为 myid。例如，有三个 zk 服务器，那么编号分别是 1,2,3。
- zxid

```
ReentranReadWriteLock 32位

高位                低位
0000000000000000  0000000000000000

                    epoch                                   xid
00000000000000000000000000000000    00000000000000000000000000000000
```

zxid 为 Long 类型，其中高 32 位表示 epoch，低 32 位表示 xid。即 zxid 由两部分构成：epoch 与 xid。
每个 Leader 都会具有一个不同的 epoch 值，表示一个时期、时代。新的 Leader 产生，则会更新所有 zkServer 的 zxid 中的 epoch。 而 xid 则为 zk 的事务 id，每一个写操作都是一个事务，都会有一个 xid。每一个写操作都需要由 Leader 发起一个提议，由所有 Follower 表决是否同意本次写操作。

- 逻辑时钟

  逻辑时钟，Logicalclock，是一个整型数，该概念在选举时称为 logicalclock，而在 zxid 中则为 epoch 的值。即 epoch 与 logicalclock 是同一个值，在不同情况下的不同名称。

- zk的选举状态

```
LOOKING，选举状态(查找 Leader 的状态)。
LEADING，领导者状态。处于该状态的服务器称为 Leader。
FOLLOWING，随从状态，同步 leader 状态。处于该状态的服务器称为 Follower。
OBSERVING，观察状态，同步 leader 状态。处于该状态的服务器称为 Observer。
```

第四集 zk集群选举发生的时机及选举算法

- 简介：zk集群选举发生的时机及选举算法
- 发生时机：整个集群群龙无首的时候（1.服务启动 2.leader宕机之后）
- 选举机制：集群中，半数zkServer同意，则产生新的leader（搭建集群时，一般都是奇数个）

  三台服务器，最多允许一台宕机，四台服务器，也是最多允许一台宕机

- 选举算法：

  对比（myid，zxid），先对比zxid，zxid大者（大表示数据越新）胜出，成为leader，如果zxid一致，则myid大者成为leader

第五集 实战--zk集群搭建

- 简介：介绍如何在linux环境中搭建zookeeper集群
- 端口的作用 2181 对client端提供服务 2888 集群内及其通讯使用的端口 3888 集群选举leader
- 修改zk配置
  - 编辑zk的conf目录下的zoo.cfg dataDir=/usr/local/zookeeper-3.4.12/data server.1=xdclass1:2888:3888 server.2=xdclass2:2888:3888 server.3=xdclass3:2888:3888

- 在zk的根目录下，新建一个data目录，并在data目录下新增一个myid的文件
- 将修改好配置的zk，分别放到三台服务器的/usr/local/,并将目录权限改为zookeeper用户
- 三台服务器，分别新增一个叫做zookeeper的用户 useradd zookeeper
- 三台服务器，均修改/usr/local/zookeeper-3.4.12/data/目录里的myid文件,文件内容是一个数字，对应server.1=xdclass1 里的1
- 三台服务器的zk的权限，都赋给zookeeper用户

  chown -R zookeeper:zookeeper zookeeper-3.4.12/
- 关闭防火墙

  systemctl stop firewalld.service
- 进入zk的bin目录

  rm -rf *.cmd

  chmod +x *.sh
- 从第一台机子依次启动 ./zkServer.sh start
- 三台机子均启动完成之后，可以使用zkServer.sh status去查看状态

第六章 分布式服务注册中心zookeeper之应用高级实战

第一集 分布式锁作用及其原理

- 简介：介绍在分布式应用中，分布式锁的作用以及其原理
- 为什么要有分布式锁

  分布式服务中，如果各个服务节点需要去竞争资源，没办法使用单机多线程中JDK自带的锁，故此时需要分布式锁来协调。
- 企业中有哪些常见的手段来实现分布式锁

  zookeeper、redis、memcache
- 分布式锁的原理

  zookeeper：去创建相应的节点，创建成功，则表示获取到相应的锁，创建失败，则表示获取锁失败

  redis、memcache:对应的去设置一个值做为锁的一标志，每次获取锁的时候，判断对应的值是否存在，存在则无法获取，不存在，则设置相应的值，表示获取到锁。（redis 使用setnx，memcache使用add）

第二集 基于zk实现分布式锁的多种方式

- 简介：介绍基于zk实现分布式锁的多种方式
- 注意事项： 创建节点的时候，一定要创建临时节点，避免应用获取到锁后，宕机，导致锁一致被持有
- 具体实现方式详见视频

第三集 实战--基于zk原生的api实现分布式锁

- 简介：介绍基于zk原生的api实现分布式锁
- 添加zk依赖

```
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.4.13</version>
</dependency>
```

- 注意点：原生api不支持递归创建节点
- 如果是单一应用，尽量不要使用分布式锁

第七章 从零开始搭建dubbo开发环境

第一集 企业中常见dubbo项目分层

- 简介：介绍在企业中常见的dubbo项目的分层
- 详见课程视频

第二集 从零开始搭建springboot+Dubbo开发环境

- 简介：从零开始搭建springboot+Dubbo开发环境
- 详见课程视频

第三集 图解dubbo开发环境搭建中各个配置的作用

- 简介：图解dubbo开发环境搭建中各个配置的作用
- 详见课程视频

第八章 dubbo核心知识讲解

第一集 dubbo核心架构及流程

- 简介：介绍面试中的高频问题点dubbo核心架构及流程
- 服务容器负责启动，加载，运行服务提供者。
- 服务提供者在启动时，向注册中心注册自己提供的服务。
- 服务消费者在启动时，向注册中心订阅自己所需的服务。
- 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。
- 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。
- 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。
- 详见 http://dubbo.apache.org/zh-cn/docs/user/preface/architecture.html

第二集 企业中dubbo常见的多种开发方式

- 简介：介绍企业中dubbo常见的多种开发方式
- github查看代码的插件 ectotree
- 通过注解的方式(多用于与springboot整合)
- 通过xml配置
- 通过api的方式----日常开发中比较少见
- 具体配置方式,详见视频

第三集 详解dubbo服务注册中心

- 简介：介绍dubbo主流的服务注册中心，包括最新集成的dubbo开源组件nacos
- simple 注册中心 ----不支持集群，不适用于生产环境

- Multicast 注册中心 ----开发阶段使用
- zookeeper 注册中心 ----目前企业中最常用，也是官方推荐
- redis 注册中心 ----虽然支持，但是较少使用
- nacos 注册中心 ----后起之秀 参考https://nacos.io/zh-cn/docs/use-nacos-with-dubbo.html

第四集 应用实战之dubbo整合zookeeper

- 简介：介绍dubbo整合zookeeper的具体步骤
- 具体整合步骤详见课程视频

第五集 应用实战之dubbo使用xml的配置方式

- 简介：介绍使用xml的配置方式进行dubbo项目的开发
- 在服务提供方，新增provider.xml，在里面配置服务提供方相应的信息
- 去掉具体服务实现类中@Service注解
- 在启动类中新增@ImportResource("provider.xml")
- 在服务消费方，新增consumer.xml的配置文件
- 在启动类中新增@ImportResource("consumer.xml")
- 使用spring的@autowire注解替代@Reference

第九章 凤凰磐涅之dubbo新特性之高级配置

第一集 高级特性之启动依赖检查

- 简介：介绍dubbo启动时的依赖检查
- Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，以便上线时，能及早发现问题，默认 check="true"。
- 可以通过 check="false" 关闭检查，比如，测试时，有些服务不关心，或者出现了循环依赖，必须有一方先启动。另外，如果你的 Spring 容器是懒加载的，或者通过 API 编程延迟引用服务，请关闭 check，否则服务临时不可用时，会抛出异常，拿到 null 引用，如果 check="false"，总是会返回引用，当服务恢复时，能自动连上。

- 具体配置的方式

  针对特定某一服务去设置 @Reference(version = "1.0.0",check = false)

  在配置文件中进行配置（实际开发中，比较建议使用配置文件的配置方式）

  启动使用 -D 参数
- 各种配置对比
  - dubbo.reference.check=false，强制改变所有 reference 的 check 值，就算配置中有声明，也会被覆盖。
  - dubbo.consumer.check=false，是设置 check 的缺省值，如果配置中有显式的声明，如：<dubbo:reference check="true"/>，不会受影响。
  - dubbo.registry.check=false，前面两个都是指订阅成功，但提供者列表是否为空是否报错，如果注册订阅失败时，也允许启动，需使用此选项，将在后台定时重试。

第二集 核心知识点之dubbo配置加载流程

- 简介：介绍dubbo配置加载流程及其配置的优先级
- 配置来源
  - JVM System Properties，-D参数
  - Externalized Configuration，外部化配置（2.7版本）
  - ServiceConfig、ReferenceConfig等编程接口采集的配置
  - 本地配置文件dubbo.properties
- 配置的优先级

  配置优先级是从上往下一次递减

  -Ddubbo.reference.com.xdclass.user.service.UserService.check=true

第三集 高级特性之dubbo超时机制及集群容错机制

- 简介：介绍dubbo的超时机制及集群容错机制
- 超时机制
  - 服务提供方进行配置
  - 服务消费方进行配置

    方法级别

    服务级别
  - 如果方法跟整个服务同时设置了超时时间，这个时候以方法设置的超时时间为准
  - 如果服务提供方跟服务消费方同时设置了超时时间，则以服务消费方为准
- 集群容错机制

当服务调用失败的时候，默认情况下，dubbo会进行重试，默认重试次数为2（不包含第一次）

通过retries="4" 设置重试次数

第四集 深入理解dubbo各协议及多协议配置

- 简介：介绍dubbo目前支持的各种协议以及如何进行多协议的配置
- 服务提供者加入相应的依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-util</artifactId>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
</dependency>

<dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-server</artifactId>
</dependency>

<dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-servlet</artifactId>
</dependency>
```

- 配置多协议

  provider.xml文件中新增想要的协议

  <dubbo:protocol name="http" port="8888"/>

  暴露服务时使用 protocol="http" 让其使用特性的协议暴露

- 修改启动类的方式为非web

  new SpringApplicationBuilder(ServiceApplication.class).web(WebApplicationType.NONE).run();

第五集 高级特性之Dubbo多注册中心及其配置

- 简介：介绍dubbo的多注册中心以及如何进行配置
- 新增<dubbo:registry address="zookeeper://192.1.1.101:2181"/>，多个注册中心使用id属性进行区分 id="remote"，服务暴露时，多个注册中心使用逗号分隔

第六集 高级特性之Dubbo服务分组及其配置

- 简介：介绍Dubbo服务分组的作用及其配置
- 当一个接口有多种实现时，可以用 group 区分。
- 服务提供者，在标签里加多个group进行区分

  <dubbo:service group="user1" interface="com.xdclass.user.service.UserService" ref="userService" />
  <dubbo:service group="user2" interface="com.xdclass.user.service.UserService" ref="userService2" />
- 服务消费者在引用的时候，也在标签里加group

  <dubbo:reference group="user2" id="userService" check="false"
  interface="com.xdclass.user.service.UserService"/>

第七集 高级特性之Dubbo服务多版本化及其配置

- 简介：介绍Dubbo服务多版本化的作用及其配置
- 当一个接口实现，出现不兼容升级时，可以用版本号过渡，版本号不同的服务相互间不引用。
- 可以按照以下的步骤进行版本迁移

  在低压力时间段，先升级一半提供者为新版本 再将所有消费者升级为新版本 然后将剩下的一半提供者升级为新版本

  一旦服务提供者制定了版本，那么所有的服务消费者也是需要去指定相应的版本
- 建议：暴露服务的时候，最好都指定下服务的版本，方便后续接口的不兼容升级

第八集 应用实战之新版管控台dubbo-ops的编译安装

- 简介：介绍新版管控台dubbo-ops的编译安装
- 旧版：dubbo-admin 新版 dubbo-ops
- 新版管理控制台主要的作用

  服务查询，服务治理(包括Dubbo2.7中新增的治理规则)以及服务测试、配置管理
- 克隆项目到本地，并编译安装和启动

  git clone https://github.com/apache/incubator-dubbo-ops.git

  cd incubator-dubbo-ops

mvn clean package

修改配置文件 dubbo-admin-server/src/main/resources/application-production.properties中的注册中心的地址

cd dubbo-distribution/target

java -jar dubbo-admin-0.1.jar

- 启动完成后，直接访问http://localhost:8080

第九集 高级特性之动态配置中心及配置外部化

- 简介：介绍动态配置中心及配置外部化
- 为什么要使用动态配置中心？
- 动态配置中心的责职

  外部化配置。启动配置的集中式存储（简单理解为dubbo.properties的外部化存储）。 服务治理。服务治理规则的存储与通知。
- 如何使用?
  - 在dubbo-ops 进行配置信息配置
  - 在项目里使用<dubbo:config-center address="zookeeper://127.0.0.1:2181"/>
- 配置中心中的配置的作用域

  如果全局配置跟应用级别均配置了相同的信息，这个时候以应用级别的为准

第十集 高级特性之元数据中心及服务测试

- 简介：介绍元数据中心的作用以及如何进行服务测试
- 作用
  - 非核心配置外部化
  - 服务测试时可以使用元数据中心的数据

第十章 课程总结

第一集 课程总结及未知知识学习方法

- 详见视频

第二集 zookeeper与dubbo核心面试题

- 简单介绍下zk选举机制？
- zk集群是怎么处理读写请求的？
- 分布式锁的实现原理以及如何实现？
- dubbo的架构是怎样的？
- Dubbo接口不兼容升级怎么处理？