参考网站：<http://www.uml.org.cn/wenzhang/artsearch.asp?curpage=1>

通过搜索关键词：Dubbo、Zookeeper等可看到很多好的学习资料。

# Dubbo

1. Dubbo是什么？

Dubbo是一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案，以及SOA服务治理方案。简单的说，dubbo就是个服务框架，如果没有分布式的需求，其实是不需要用的，只有在分布式的时候，才有dubbo这样的分布式服务框架的需求，并且本质上是个服务调用的东东，说白了就是个远程服务调用的分布式框架（告别Web Service模式中的WSdl，以服务者与消费者的方式在dubbo上注册）

其核心部分包含:

1. 远程通讯: 提供对多种基于长连接的NIO框架抽象封装，包括多种线程模型，序列化，以及“请求-响应”模式的信息交换方式。

2. 集群容错: 提供基于接口方法的透明远程过程调用，包括多协议支持，以及软负载均衡，失败容错，地址路由，动态配置等集群支持。

3. 自动发现: 基于注册中心目录服务，使服务消费方能动态的查找服务提供方，使地址透明，使服务提供方可以平滑增加或减少机器。

2. Dubbo能做什么？

1.透明化的远程方法调用，就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。

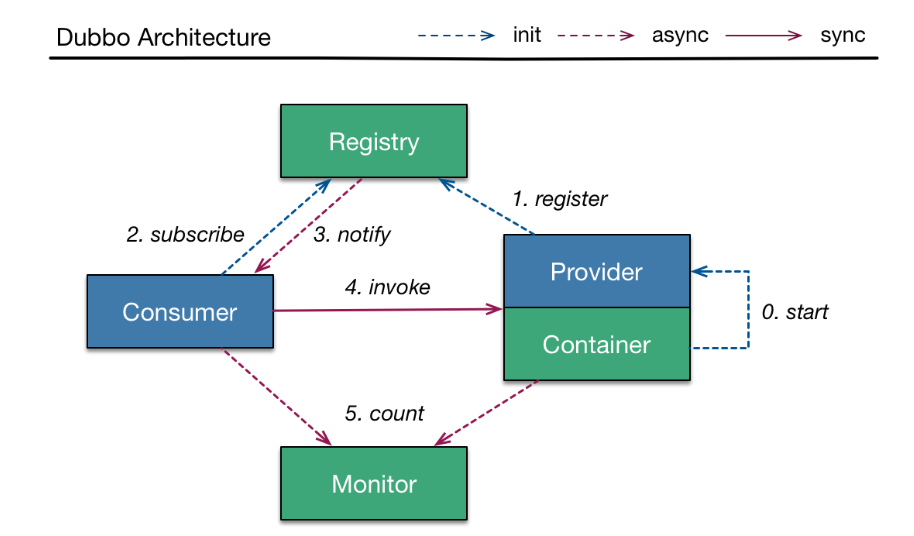
2.软负载均衡及容错机制，可在内网替代F5等硬件负载均衡器，降低成本，减少单点。

3. 服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。

Dubbo采用全Spring配置方式，透明化接入应用，对应用没有任何API侵入，只需用Spring加载Dubbo的配置即可，Dubbo基于Spring的Schema扩展进行加载。

之前使用Web Service，我想测试接口可以通过模拟消息的方式通过soapui或LR进行功能测试或性能测试。但现在使用Dubbo，接口之间不能直接交互，我尝试通过模拟消费者地址测试，结果不堪入目，再而使用jmeter通过junit进行测试，但还是需要往dubbo上去注册，如果再不给提供源代码的前提下，这个测试用例不好写啊....

Dubbo |ˈdʌbəʊ| is a high-performance, java based RPC framework open-sourced by Alibaba. As in many RPC systems, dubbo is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. On the server side, the server implements this interface and runs a dubbo server to handle client calls. On the client side, the client has a stub that provides the same methods as the server.



Dubbo offers three key functionalities, which include interface based remote call, fault tolerance & load balancing, and automatic service registration & discovery. Dubbo framework is widely adopted inside Alibaba and outside by other companies including jingdong, dangdang, qunar, kaola, and many others.

## 节点角色说明

| **节点** | **角色说明** |
| --- | --- |
| 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 |

## 调用关系说明

1. 服务容器负责启动，加载，运行服务提供者。
2. 服务提供者在启动时，向注册中心注册自己提供的服务。
3. 服务消费者在启动时，向注册中心订阅自己所需的服务。
4. 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。
5. 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。
6. 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。

Dubbo 架构具有以下几个特点，分别是连通性、健壮性、伸缩性、以及向未来架构的升级性。

## 连通性

* 注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小
* 监控中心负责统计各服务调用次数，调用时间等，统计先在内存汇总后每分钟一次发送到监控中心服务器，并以报表展示
* 服务提供者向注册中心注册其提供的服务，并汇报调用时间到监控中心，此时间不包含网络开销
* 服务消费者向注册中心获取服务提供者地址列表，并根据负载算法直接调用提供者，同时汇报调用时间到监控中心，此时间包含网络开销
* 注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外
* 注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者
* 注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表
* 注册中心和监控中心都是可选的，服务消费者可以直连服务提供者

## 健状性

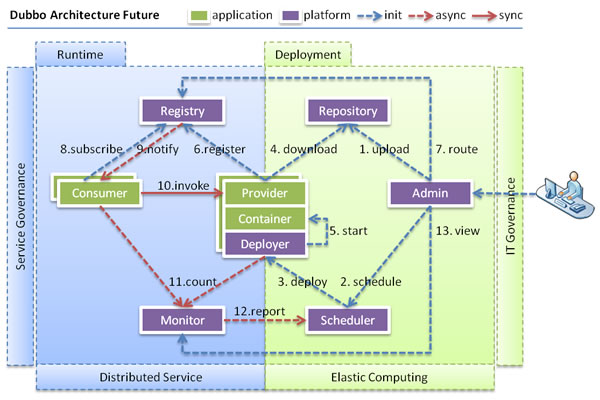
* 监控中心宕掉不影响使用，只是丢失部分采样数据
* 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
* 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
* 注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
* 服务提供者无状态，任意一台宕掉后，不影响使用
* 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

## 伸缩性

* 注册中心为对等集群，可动态增加机器部署实例，所有客户端将自动发现新的注册中心
* 服务提供者无状态，可动态增加机器部署实例，注册中心将推送新的服务提供者信息给消费者

## 升级性

当服务集群规模进一步扩大，带动IT治理结构进一步升级，需要实现动态部署，进行流动计算，现有分布式服务架构不会带来阻力。下图是未来可能的一种架构：



## 节点角色说明

| **节点** | **角色说明** |
| --- | --- |
| Deployer | 自动部署服务的本地代理 |
| Repository | 仓库用于存储服务应用发布包 |
| Scheduler | 调度中心基于访问压力自动增减服务提供者 |
| Admin | 统一管理控制台 |
| 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 |

# Dubbo文档（阿里巴巴根据官方文档）

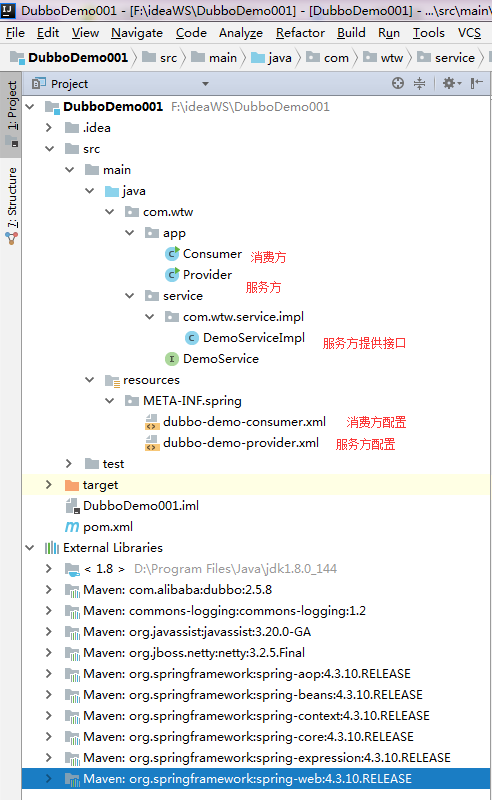


http长连接：<https://www.cnblogs.com/cswuyg/p/3653263.html>

# 二、入门案例（按照官网搭建：<http://dubbo.io/>）

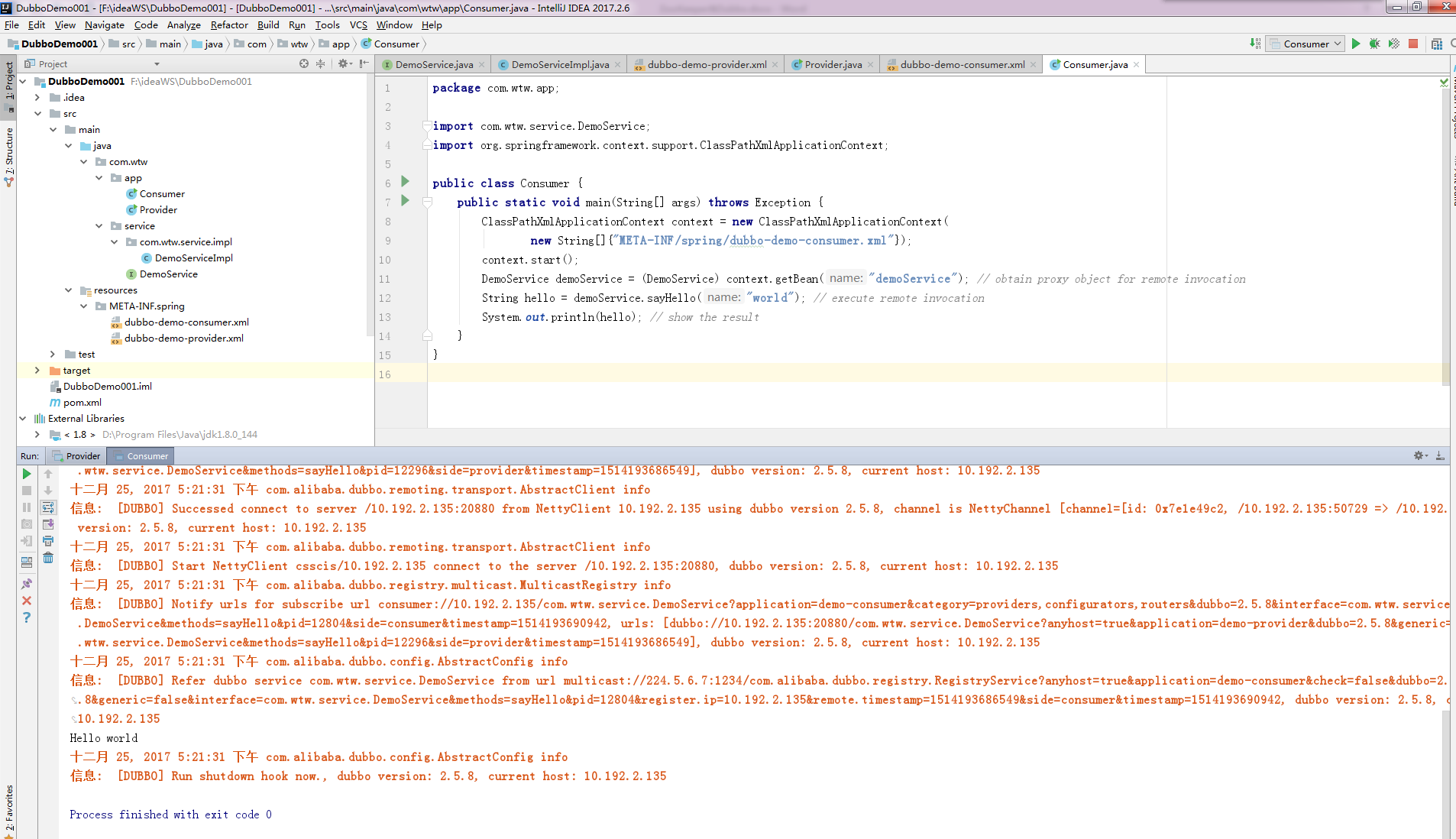
该案例只有一个服务提供方、一个消费者和一个Multicast注册中心（用于测试）。

项目目录结构：



使用时需要先开启服务端，再开启客户端，运行结果如图：







# 三、Dubbo+zookeeper集群与负载均衡

Zookeeper作为Dubbo服务的注册中心，Dubbo原先基于数据库的注册中心，没采用Zookeeper，Zookeeper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，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。

参考：http://blog.csdn.net/IRhythm/article/details/51340862

# 四、Dubbo应用场景

官网上的说话都太笼统，现在缺乏这方面经验，还没接触过它的实际应用案例。

案例一：

dubbo+spring boot+mybatis+redis等主流技术搭建成一套分布式服务框架

链接：<https://github.com/dearbinge/dubbo-spring-boot-mybatis-redis>

案例二：

Spring Boot+dubbo+docker

链接：<https://github.com/linux-china/spring-boot-dubbo>

# 五、Zookeeper

ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，它包含一个简单的原语集，分布式应用程序可以基于它实现同步服务，配置维护和命名服务等。Zookeeper是hadoop的一个子项目，其发展历程无需赘述。在分布式应用中，由于工程师不能很好地使用锁机制，以及基于消息的协调机制不适合在某些应用中使用，因此需要有一种可靠的、可扩展的、分布式的、可配置的协调机制来统一系统的状态。Zookeeper的目的就在于此。本文简单分析zookeeper的工作原理，对于如何使用zookeeper不是本文讨论的重点。

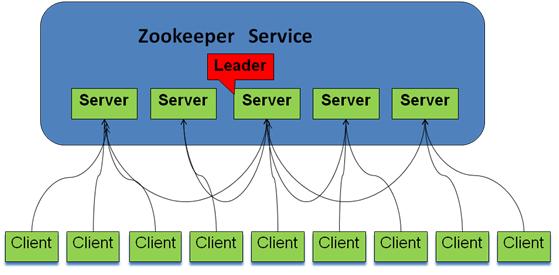
**1 Zookeeper的基本概念**

**1.1 角色**

Zookeeper中的角色主要有以下三类，如下表所示：



系统模型如图所示：



**2 ZooKeeper的工作原理**

Zookeeper的核心是原子广播，这个机制保证了各个Server之间的同步。实现这个机制的协议叫做Zab协议。Zab协议有两种模式，它们分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，Zab就进入了恢复模式，当领导者被选举出来，且大多数Server完成了和leader的状态同步以后，恢复模式就结束了。状态同步保证了leader和Server具有相同的系统状态。

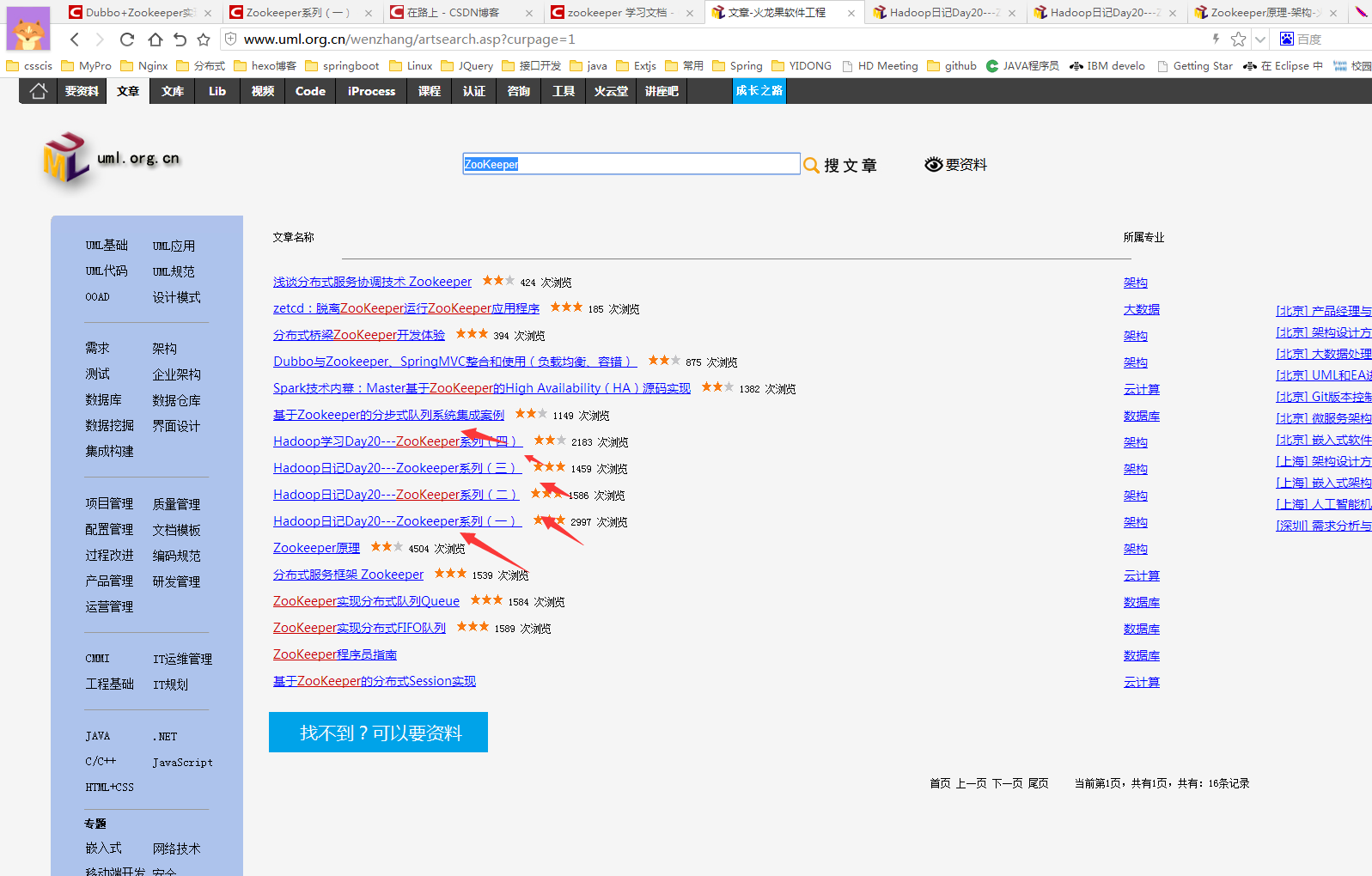
**选主流程、 同步流程、工作流程**

详细如下：

ZooKeeper学习参考如下链接：

<http://www.uml.org.cn/wenzhang/artsearch.asp?curpage=1>

搜索：ZooKeeper



zookeeper是什么：

Zookeeper,一种分布式应用的协作服务,是Google的Chubby一个开源的实现,是Hadoop的分布式协调服务,它包含一个简单的原语集,应用于分布式应用的协作服务,使得分布式应用可以基于这些接口实现诸如同步、配置维护和分集群或者命名的服务。

zookeeper是一个由多个service组成的集群,一个leader,多个follower,每个server保存一份数据部分,全局数据一致,分布式读写,更新请求转发由leader实施.

更新请求顺序进行,来自同一个client的更新请求按其发送顺序依次执行,数据更新原子性,一次数据更新要么成功,要么失败,全局唯一数据试图,client无论连接到哪个server,数据试图是一致的.

为什么要用zookeeper

大部分分布式应用需要一个主控、协调器或控制器来管理物理分布的子进程（如资源、任务分配等）,目前,大部分应用需要开发私有的协调程序,缺乏一个通用的机制.协调程序的反复编写浪费,且难以形成通用、伸缩性好的协调器,ZooKeeper：提供通用的分布式锁服务,用以协调分布式应用

zookeeper工作原理

zookeeper的核心是原子广播,这个机制保证了各个server之间的同步,实现这个机制的协议叫做Zab协议.Zab协议有两种模式,他们分别是恢复模式和广播模式.

　 1.当服务启动或者在领导者崩溃后,Zab就进入了恢复模式,当领导着被选举出来,且大多数server都完成了和leader的状态同步后,恢复模式就结束了.状态同步保证了leader和server具有相同的系统状态.

　 2.一旦leader已经和多数的follower进行了状态同步后,他就可以开始广播消息了,即进入广播状态.这时候当一个server加入zookeeper服务中,它会在恢复模式下启动,发下leader,并和leader进行状态同步,待到同步结束,它也参与广播消息.

说明:

广播模式需要保证proposal被按顺序处理,因此zk采用了递增的事务id号(zxid)来保证.所有的提议(proposal)都在被提出的时候加上了zxid.实现中zxid是一个64为的数字,它高32位是epoch用来标识leader关系是否改变,每次一个leader被选出来,它都会有一个新的epoch.低32位是个递增计数.

当leader崩溃或者leader失去大多数的follower,这时候zk进入恢复模式,恢复模式需要重新选举出一个新的leader,让所有的server都恢复到一个正确的状态.

zookeeper服务一致维持在Broadcast状态,直到leader崩溃了或者leader失去了大部分的followers支持.

Broadcast模式极其类似于分布式事务中的2pc（two-phrase commit两阶段提交）：即leader提起一个决议,由followers进行投票,leader对投票结果进行计算决定是否通过该决议,如果通过执行该决议（事务）,否则什么也不做.

Leader选举

每个Server启动以后都询问其它的Server它要投票给谁,对于其他server的询问,server每次根据自己的状态都回复自己推荐的leader的id和上一次处理事务的zxid（系统启动时每个server都会推荐自己）,收到所有Server回复以后,就计算出zxid最大的哪个Server,并将这个Server相关信息设置成下一次要投票的Server.计算这过程中获得票数最多的的sever为获胜者,如果获胜者的票数超过半数,则改server被选为leader.否则,继续这个过程,直到leader被选举出来.leader就会开始等待server连接,Follower连接leader,将最大的zxid发送给leader,Leader根据follower的zxid确定同步点,完成同步后通知follower已经成为uptodate状态,Follower收到uptodate消息后,又可以重新接受client的请求进行服务了.

zookeeper的数据模型

层次化的目录结构,命名符合常规文件系统规范

每个节点在zookeeper中叫做znode,并且其有一个唯一的路径标识

节点Znode可以包含数据和子节点,但是EPHEMERAL类型的节点不能有子节点

Znode中的数据可以有多个版本,比如某一个路径下存有多个数据版本,那么查询这个路径下的数据就需要带上版本

客户端应用可以在节点上设置监视器,节点不支持部分读写,而是一次性完整读写

Zoopkeeper 提供了一套很好的分布式集群管理的机制,就是它这种基于层次型的目录树的数据结构,并对树中的节点进行有效管理,从而可以设计出多种多样的分布式的数据管理模型

Zookeeper的节点

Znode有两种类型,短暂的（ephemeral）和持久的（persistent）

Znode的类型在创建时确定并且之后不能再修改

短暂znode的客户端会话结束时,zookeeper会将该短暂znode删除,短暂znode不可以有子节点

持久znode不依赖于客户端会话,只有当客户端明确要删除该持久znode时才会被删除

Znode有四种形式的目录节点,PERSISTENT、PERSISTENT\_SEQUENTIAL、EPHEMERAL、EPHEMERAL\_SEQUENTIAL.

znode 可以被监控,包括这个目录节点中存储的数据的修改,子节点目录的变化等,一旦变化可以通知设置监控的客户端,这个功能是zookeeper对于应用最重要的特性,

通过这个特性可以实现的功能包括配置的集中管理,集群管理,分布式锁等等.

Zookeeper的角色

领导者（leader）,负责进行投票的发起和决议,更新系统状态

学习者（learner）,包括跟随者（follower）和观察者（observer）.

follower用于接受客户端请求并想客户端返回结果,在选主过程中参与投票

Observer可以接受客户端连接,将写请求转发给leader,但observer不参加投票过程,只同步leader的状态,observer的目的是为了扩展系统,提高读取速度

客户端（client）,请求发起方

Watcher

Watcher 在 ZooKeeper是一个核心功能,Watcher可以监控目录节点的数据变化以及子目录的变化,一旦这些状态发生变化,服务器就会通知所有设置在这个目录节点上的Watcher,从而每个客户端都很快知道它所关注的目录节点的状态发生变化,而做出相应的反应

可以设置观察的操作：exists,getChildren,getData

可以触发观察的操作：create,delete,setData

znode以某种方式发生变化时,“观察”（watch）机制可以让客户端得到通知.

可以针对ZooKeeper服务的“操作”来设置观察,该服务的其他 操作可以触发观察.

比如,客户端可以对某个客户端调用exists操作,同时在它上面设置一个观察,如果此时这个znode不存在,则exists返回false,如果一段时间之后,这个znode被其他客户端创建,则这个观察会被触发,之前的那个客户端就会得到通知.

Zookeeper集群搭建

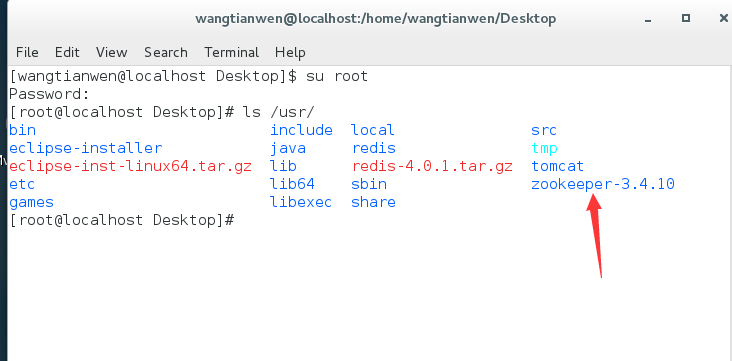
Zookeeper 不仅可以单机提供服务,同时也支持多机组成集群来提供服务,实际上Zookeeper还支持另外一种伪集群的方式,也就是可以在一台物理机上运行多个Zookeeper实例.

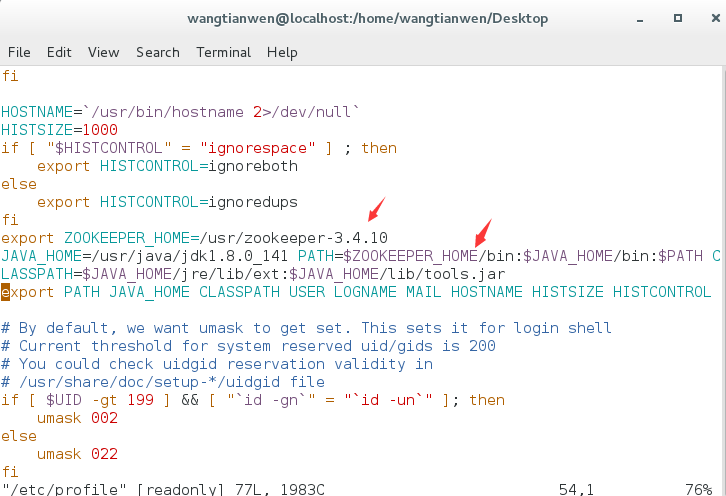
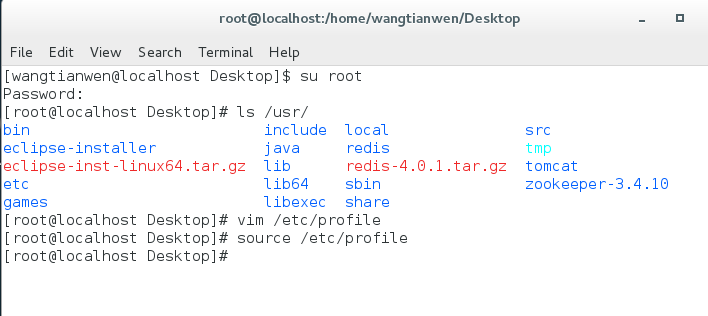
Zookeeper通过复制来实现高可用性,只要集合体中半数以上的机器处于可用状态,它就能够保证服务继续。

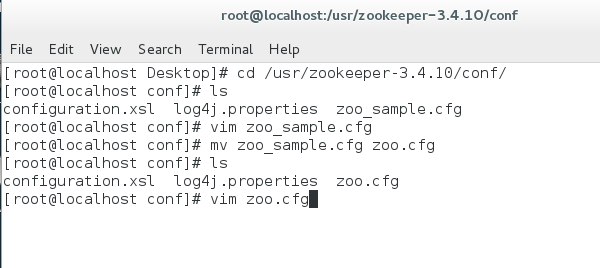
集群容灾性:

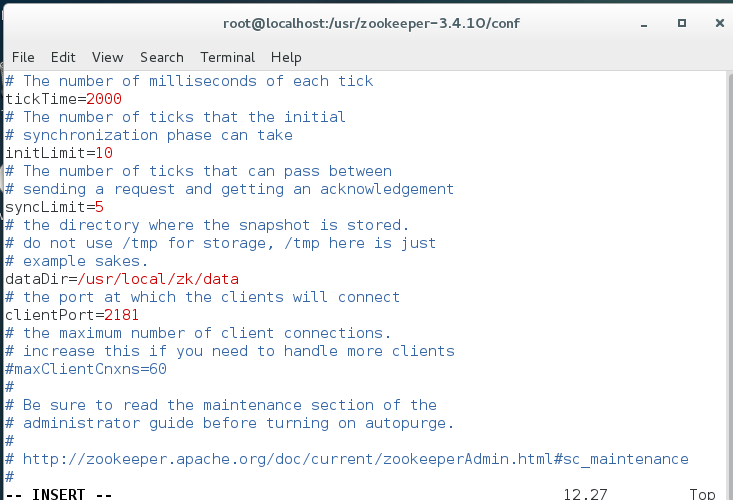
　3台机器只要有2台可用就可以选出leader并且对外提供服务(2n+1台机器,可以容n台机器挂掉)。

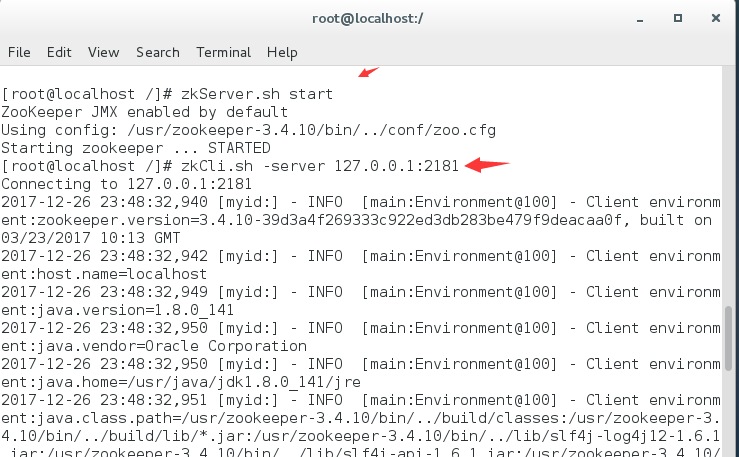
## 自己搭建zookeeper Linux 单机环境

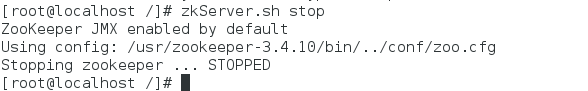












## Zookeeper的伪集群模式搭建

参考：<http://www.uml.org.cn/zjjs/201411171.asp?artid=15501>

## Zookeeper的集群模式搭建

参考：<http://www.uml.org.cn/zjjs/201411171.asp?artid=15501>

# 六、Docker

# 七、Hadoop