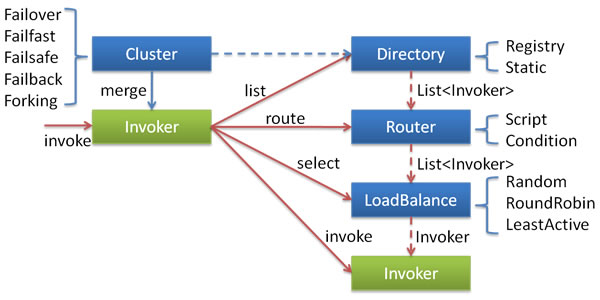
# 基于dubbo创建分布式服务

Dubbo是Alibaba开源的分布式服务框架，我们可以非常容易地通过Dubbo来构建分布式服务，并根据自己实际业务应用场景来选择合适的集群容错模式，这个对于很多应用都是迫切希望的，只需要通过简单的配置就能够实现分布式服务调用，也就是说服务提供方（Provider）发布的服务可以天然就是集群服务，比如，在实时性要求很高的应用场景下，可能希望来自消费方（Consumer）的调用响应时间最短，只需要选择Dubbo的Forking Cluster模式配置，就可以对一个调用请求并行发送到多台对等的提供方（Provider）服务所在的节点上，只选择最快一个返回响应的，然后将调用结果返回给服务消费方（Consumer），显然这种方式是以冗余服务为基础的，需要消耗更多的资源，但是能够满足高实时应用的需求。  
这里主要围绕Dubbo分布式服务相关配置的使用来说明与实践。

**Dubbo服务集群容错**

假设我们使用的是单机模式的Dubbo服务，如果在服务提供方（Provider）发布服务以后，服务消费方（Consumer）发出一次调用请求，恰好这次由于网络问题调用失败，那么我们可以配置服务消费方重试策略，可能消费方第二次重试调用是成功的（重试策略只需要配置即可，重试过程是透明的）；但是，如果服务提供方发布服务所在的节点发生故障，那么消费方再怎么重试调用都是失败的，所以我们需要采用集群容错模式，这样如果单个服务节点因故障无法提供服务，还可以根据配置的集群容错模式，调用其他可用的服务节点，这就提高了服务的可用性。  
首先，根据Dubbo文档，我们引用文档提供的一个架构图以及各组件关系说明，如下所示：  
  
上述各个组件之间的关系（引自Dubbo文档）说明如下：



* 这里的Invoker是Provider的一个可调用Service的抽象，Invoker封装了Provider地址及Service接口信息。
* Directory代表多个Invoker，可以把它看成List，但与List不同的是，它的值可能是动态变化的，比如注册中心推送变更。
* Cluster将Directory中的多个Invoker伪装成一个Invoker，对上层透明，伪装过程包含了容错逻辑，调用失败后，重试另一个。
* Router负责从多个Invoker中按路由规则选出子集，比如读写分离，应用隔离等。
* LoadBalance负责从多个Invoker中选出具体的一个用于本次调用，选的过程包含了负载均衡算法，调用失败后，需要重选。

我们也简单说明目前Dubbo支持的集群容错模式，每种模式适应特定的应用场景，可以根据实际需要进行选择。Dubbo内置支持如下6种集群模式：

* Failover Cluster模式

配置值为failover。这种模式是Dubbo集群容错默认的模式选择，调用失败时，会自动切换，重新尝试调用其他节点上可用的服务。对于一些幂等性操作可以使用该模式，如读操作，因为每次调用的副作用是相同的，所以可以选择自动切换并重试调用，对调用者完全透明。可以看到，如果重试调用必然会带来响应端的延迟，如果出现大量的重试调用，可能说明我们的服务提供方发布的服务有问题，如网络延迟严重、硬件设备需要升级、程序算法非常耗时，等等，这就需要仔细检测排查了。  
例如，可以这样显式指定Failover模式，或者不配置则默认开启Failover模式，配置示例如下：

|  |  |
| --- | --- |
| 1 | <[dubbo:service](http://dubboservice) interface="org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService"version="1.0.0" |

|  |  |
| --- | --- |
| 2 | cluster="failover" retries="2" timeout="100"ref="chatRoomOnlineUserCounterService" protocol="dubbo" > |

|  |  |
| --- | --- |
| 3 | <[dubbo:method](http://dubbomethod) name="queryRoomUserCount" timeout="80" retries="2" /> |

|  |  |
| --- | --- |
| 4 | </[dubbo:service](http://dubboservice)> |

上述配置使用Failover Cluster模式，如果调用失败一次，可以再次重试2次调用，服务级别调用超时时间为100ms，调用方法queryRoomUserCount的超时时间为80ms，允许重试2次，最坏情况调用花费时间160ms。如果该服务接口org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService还有其他的方法可供调用，则其他方法没有显式配置则会继承使用dubbo:service配置的属性值。

* Failfast Cluster模式

配置值为failfast。这种模式称为快速失败模式，调用只执行一次，失败则立即报错。这种模式适用于非幂等性操作，每次调用的副作用是不同的，如写操作，比如交易系统我们要下订单，如果一次失败就应该让它失败，通常由服务消费方控制是否重新发起下订单操作请求（另一个新的订单）。

* Failsafe Cluster模式

配置值为failsafe。失败安全模式，如果调用失败， 则直接忽略失败的调用，而是要记录下失败的调用到日志文件，以便后续审计。

* Failback Cluster模式

配置值为failback。失败自动恢复，后台记录失败请求，定时重发。通常用于消息通知操作。

* Forking Cluster模式

配置值为forking。并行调用多个服务器，只要一个成功即返回。通常用于实时性要求较高的读操作，但需要浪费更多服务资源。

* Broadcast Cluster模式

配置值为broadcast。广播调用所有提供者，逐个调用，任意一台报错则报错（2.1.0开始支持）。通常用于通知所有提供者更新缓存或日志等本地资源信息。  
上面的6种模式都可以应用于生产环境，我们可以根据实际应用场景选择合适的集群容错模式。如果我们觉得Dubbo内置提供的几种集群容错模式都不能满足应用需要，也可以定制实现自己的集群容错模式，因为Dubbo框架给我提供的扩展的接口，只需要实现接口com.alibaba.dubbo.rpc.cluster.Cluster即可，接口定义如下所示：

|  |  |
| --- | --- |
| 01 | @SPI(FailoverCluster.NAME) |

|  |  |
| --- | --- |
| 02 | public interface Cluster { |

|  |  |
| --- | --- |
| 03 |  |

|  |  |
| --- | --- |
| 04 | /\*\* |

|  |  |
| --- | --- |
| 05 | \* Merge the directory invokers to a virtual invoker. |

|  |  |
| --- | --- |
| 06 | \* @param <T> |

|  |  |
| --- | --- |
| 07 | \* @param directory |

|  |  |
| --- | --- |
| 08 | \* @return cluster invoker |

|  |  |
| --- | --- |
| 09 | \* @throws RpcException |

|  |  |
| --- | --- |
| 10 | \*/ |

|  |  |
| --- | --- |
| 11 | @Adaptive |

|  |  |
| --- | --- |
| 12 | <T> Invoker<T> join(Directory<T> directory) throws RpcException; |

|  |  |
| --- | --- |
| 13 |  |

|  |  |
| --- | --- |
| 14 | } |

关于如何实现一个自定义的集群容错模式，可以参考Dubbo源码中内置支持的汲取你容错模式的实现，6种模式对应的实现类如下所示：

|  |  |
| --- | --- |
| 1 | com.alibaba.dubbo.rpc.cluster.support.FailoverCluster |

|  |  |
| --- | --- |
| 2 | com.alibaba.dubbo.rpc.cluster.support.FailfastCluster |

|  |  |
| --- | --- |
| 3 | com.alibaba.dubbo.rpc.cluster.support.FailsafeCluster |

|  |  |
| --- | --- |
| 4 | com.alibaba.dubbo.rpc.cluster.support.FailbackCluster |

|  |  |
| --- | --- |
| 5 | com.alibaba.dubbo.rpc.cluster.support.ForkingCluster |

|  |  |
| --- | --- |
| 6 | com.alibaba.dubbo.rpc.cluster.support.AvailableCluster |

可能我们初次接触Dubbo时，不知道如何在实际开发过程中使用Dubbo的集群模式，后面我们会以Failover Cluster模式为例开发我们的分布式应用，再进行详细的介绍。

**Dubbo服务负载均衡**

Dubbo框架内置提供负载均衡的功能以及扩展接口，我们可以透明地扩展一个服务或服务集群，根据需要非常容易地增加/移除节点，提高服务的可伸缩性。Dubbo框架内置提供了4种负载均衡策略，如下所示：

* Random LoadBalance：随机策略，配置值为random。可以设置权重，有利于充分利用服务器的资源，高配的可以设置权重大一些，低配的可以稍微小一些
* RoundRobin LoadBalance：轮询策略，配置值为roundrobin。
* LeastActive LoadBalance：配置值为leastactive。根据请求调用的次数计数，处理请求更慢的节点会受到更少的请求
* ConsistentHash LoadBalance：一致性Hash策略，具体配置方法可以参考Dubbo文档。相同调用参数的请求会发送到同一个服务提供方节点上，如果某个节点发生故障无法提供服务，则会基于一致性Hash算法映射到虚拟节点上（其他服务提供方）

在实际使用中，只需要选择合适的负载均衡策略值，配置即可，下面是上述四种负载均衡策略配置的示例：

|  |  |
| --- | --- |
| 1 | <[dubbo:service](http://dubboservice) interface="org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService"version="1.0.0" |

|  |  |
| --- | --- |
| 2 | cluster="failover" retries="2" timeout="100" loadbalance="random" |

|  |  |
| --- | --- |
| 3 | ref="chatRoomOnlineUserCounterService" protocol="dubbo" > |

|  |  |
| --- | --- |
| 4 | <[dubbo:method](http://dubbomethod) name="queryRoomUserCount" timeout="80" retries="2"loadbalance="leastactive" /> |

|  |  |
| --- | --- |
| 5 | </[dubbo:service](http://dubboservice)> |

上述配置，也体现了Dubbo配置的继承性特点，也就是dubbo:service元素配置了loadbalance=”random”，则该元素的子元素dubbo:method如果没有指定负载均衡策略，则默认为loadbalance=”random”，否则如果dubbo:method指定了loadbalance=”leastactive”，则使用子元素配置的负载均衡策略覆盖了父元素指定的策略（这里调用queryRoomUserCount方法使用leastactive负载均衡策略）。

当然，Dubbo框架也提供了实现自定义负载均衡策略的接口，可以实现com.alibaba.dubbo.rpc.cluster.LoadBalance接口，接口定义如下所示：

|  |  |
| --- | --- |
| 01 | /\*\* |

|  |  |
| --- | --- |
| 02 | \* LoadBalance. (SPI, Singleton, ThreadSafe) |

|  |  |
| --- | --- |
| 03 | \* |

|  |  |
| --- | --- |
| 04 | \* <a href="<http://en.wikipedia.org/wiki/Load_balancing_>(computing)">Load-Balancing</a> |

|  |  |
| --- | --- |
| 05 | \* |

|  |  |
| --- | --- |
| 06 | \* @see com.alibaba.dubbo.rpc.cluster.Cluster#join(Directory) |

|  |  |
| --- | --- |
| 07 | \* @author qian.lei |

|  |  |
| --- | --- |
| 08 | \* @author william.liangf |

|  |  |
| --- | --- |
| 09 | \*/ |

|  |  |
| --- | --- |
| 10 | @SPI(RandomLoadBalance.NAME) |

|  |  |
| --- | --- |
| 11 | public interface LoadBalance { |

|  |  |
| --- | --- |
| 12 |  |

|  |  |
| --- | --- |
| 13 | /\*\* |

|  |  |
| --- | --- |
| 14 | \* select one invoker in list. |

|  |  |
| --- | --- |
| 15 | \* @param invokers invokers. |

|  |  |
| --- | --- |
| 16 | \* @param url refer url |

|  |  |
| --- | --- |
| 17 | \* @param invocation invocation. |

|  |  |
| --- | --- |
| 18 | \* @return selected invoker. |

|  |  |
| --- | --- |
| 19 | \*/ |

|  |  |
| --- | --- |
| 20 | @Adaptive("loadbalance") |

|  |  |
| --- | --- |
| 21 | <T> Invoker<T> select(List<Invoker<T>> invokers, URL url, Invocation invocation)throws RpcException; |

|  |  |
| --- | --- |
| 22 |  |

|  |  |
| --- | --- |
| 23 | } |

如何实现一个自定义负载均衡策略，可以参考Dubbo框架内置的实现，如下所示的3个实现类：

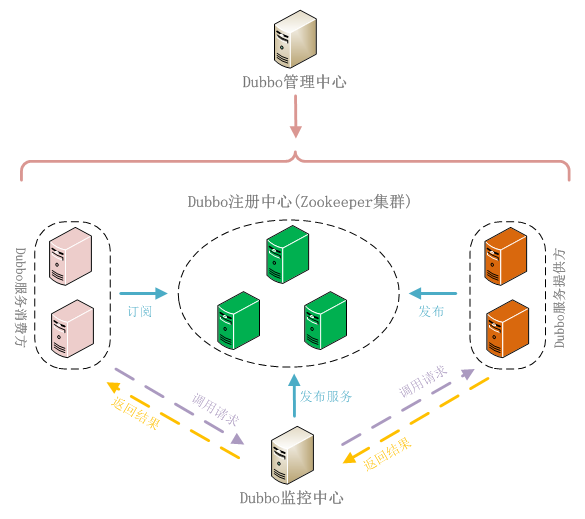
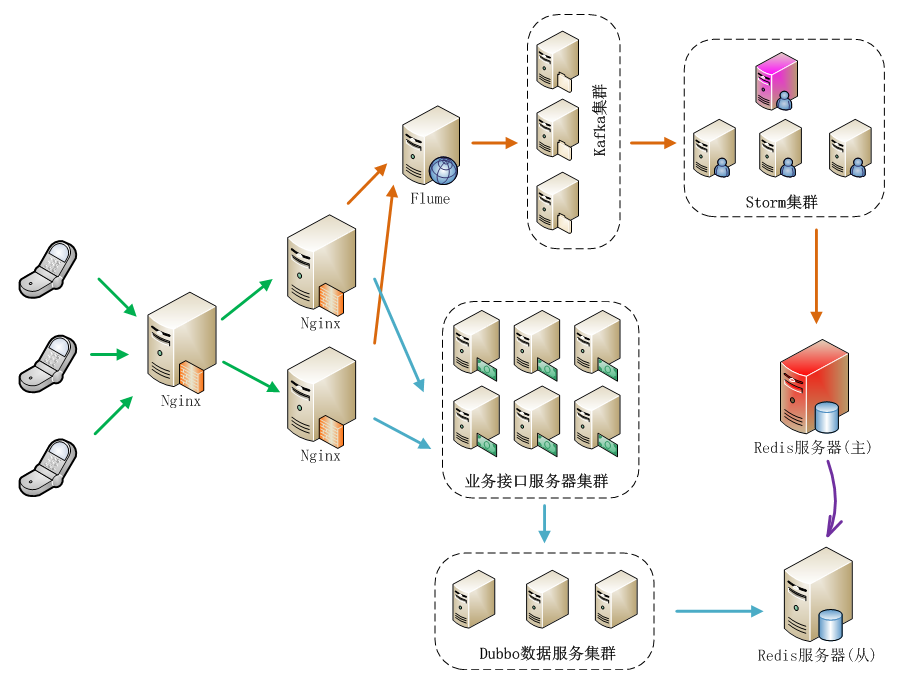
|  |  |
| --- | --- |
| 1 | com.alibaba.dubbo.rpc.cluster.loadbalance.RandomLoadBalance |

|  |  |
| --- | --- |
| 2 | com.alibaba.dubbo.rpc.cluster.loadbalance.RoundRobinLoadBalance |

|  |  |
| --- | --- |
| 3 | com.alibaba.dubbo.rpc.cluster.loadbalance.LeastActiveLoadBalance |

**Dubbo服务集群容错实践**

手机应用是以聊天室为基础的，我们需要收集用户的操作行为，然后计算聊天室中在线人数，并实时在手机应用端显示人数，整个系统的架构如图所示：  
  
上图中，主要包括了两大主要流程：日志收集并实时处理流程、调用读取实时计算结果流程，我们使用基于Dubbo框架开发的服务来提供实时计算结果读取聊天人数的功能。上图中，实际上业务接口服务器集群也可以基于Dubbo框架构建服务，就看我们想要构建什么样的系统来满足我们的需要。  
如果不使用注册中心，服务消费方也能够直接调用服务提供方发布的服务，这样需要服务提供方将服务地址暴露给服务消费方，而且也无法使用监控中心的功能，这种方式成为直连。  
如果我们使用注册中心，服务提供方将服务发布到注册中心，而服务消费方可以通过注册中心订阅服务，接收服务提供方服务变更通知，这种方式可以隐藏服务提供方的细节，包括服务器地址等敏感信息，而服务消费方只能通过注册中心来获取到已注册的提供方服务，而不能直接跨过注册中心与服务提供方直接连接。这种方式的好处是还可以使用监控中心服务，能够对服务的调用情况进行监控分析，还能使用Dubbo服务管理中心，方便管理服务，我们在这里使用的是这种方式，也推荐使用这种方式。使用注册中心的Dubbo分布式服务相关组件结构，如下图所示：



下面，开发部署我们的应用，通过如下4个步骤来完成：

* 服务接口定义

服务接口将服务提供方（Provider）和服务消费方（Consumer）连接起来，服务提供方实现接口中定义的服务，即给出服务的实现，而服务消费方负责调用服务。我们接口中给出了2个方法，一个是实时查询获取当前聊天室内人数，另一个是查询一天中某个/某些聊天室中在线人数峰值，接口定义如下所示：

|  |  |
| --- | --- |
| 01 | package org.shirdrn.dubbo.api; |

|  |  |
| --- | --- |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.List; |

|  |  |
| --- | --- |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public interface ChatRoomOnlineUserCounterService { |

|  |  |
| --- | --- |
| 06 |  |

|  |  |
| --- | --- |
| 07 | String queryRoomUserCount(String rooms); |

|  |  |
| --- | --- |
| 08 |  |

|  |  |
| --- | --- |
| 09 | List<String> getMaxOnlineUserCount(List<String> rooms, String date, String dateFormat); |

|  |  |
| --- | --- |
| 10 | } |

接口是服务提供方和服务消费方公共遵守的协议，一般情况下是服务提供方将接口定义好后提供给服务消费方。

* 服务提供方

服务提供方实现接口中定义的服务，其实现和普通的服务没什么区别，我们的实现类为ChatRoomOnlineUserCounterServiceImpl，代码如下所示：

|  |  |
| --- | --- |
| 01 | package org.shirdrn.dubbo.provider.service; |

|  |  |
| --- | --- |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.List; |

|  |  |
| --- | --- |
| 04 |  |

|  |  |
| --- | --- |
| 05 | import org.apache.commons.logging.Log; |

|  |  |
| --- | --- |
| 06 | import org.apache.commons.logging.LogFactory; |

|  |  |
| --- | --- |
| 07 | import org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService; |

|  |  |
| --- | --- |
| 08 | import org.shirdrn.dubbo.common.utils.DateTimeUtils; |

|  |  |
| --- | --- |
| 09 |  |

|  |  |
| --- | --- |
| 10 | import redis.clients.jedis.Jedis; |

|  |  |
| --- | --- |
| 11 | import redis.clients.jedis.JedisPool; |

|  |  |
| --- | --- |
| 12 |  |

|  |  |
| --- | --- |
| 13 | import com.alibaba.dubbo.common.utils.StringUtils; |

|  |  |
| --- | --- |
| 14 | import com.google.common.base.Strings; |

|  |  |
| --- | --- |
| 15 | import com.google.common.collect.Lists; |

|  |  |
| --- | --- |
| 16 |  |

|  |  |
| --- | --- |
| 17 | public class ChatRoomOnlineUserCounterServiceImpl implementsChatRoomOnlineUserCounterService { |

|  |  |
| --- | --- |
| 18 |  |

|  |  |
| --- | --- |
| 19 | private static final Log LOG = LogFactory.getLog(ChatRoomOnlineUserCounterServiceImpl.class); |

|  |  |
| --- | --- |
| 20 | private JedisPool jedisPool; |

|  |  |
| --- | --- |
| 21 | private static final String KEY\_USER\_COUNT = "chat::room::play::user::cnt"; |

|  |  |
| --- | --- |
| 22 | private static final String KEY\_MAX\_USER\_COUNT\_PREFIX ="chat::room::max::user::cnt::"; |

|  |  |
| --- | --- |
| 23 | private static final String DF\_YYYYMMDD = "yyyyMMdd"; |

|  |  |
| --- | --- |
| 24 |  |

|  |  |
| --- | --- |
| 25 | public String queryRoomUserCount(String rooms) { |

|  |  |
| --- | --- |
| 26 | LOG.info("Params[Server|Recv|REQ] rooms=" + rooms); |

|  |  |
| --- | --- |
| 27 | StringBuffer builder = new StringBuffer(); |

|  |  |
| --- | --- |
| 28 | if(!Strings.isNullOrEmpty(rooms)) { |

|  |  |
| --- | --- |
| 29 | Jedis jedis = null; |

|  |  |
| --- | --- |
| 30 | try { |

|  |  |
| --- | --- |
| 31 | jedis = jedisPool.getResource(); |

|  |  |
| --- | --- |
| 32 | String[] fields = rooms.split(","); |

|  |  |
| --- | --- |
| 33 | List<String> results = jedis.hmget(KEY\_USER\_COUNT, fields); |

|  |  |
| --- | --- |
| 34 | builder.append(StringUtils.join(results, ",")); |

|  |  |
| --- | --- |
| 35 | } catch (Exception e) { |

|  |  |
| --- | --- |
| 36 | LOG.error("", e); |

|  |  |
| --- | --- |
| 37 | } finally { |

|  |  |
| --- | --- |
| 38 | if(jedis != null) { |

|  |  |
| --- | --- |
| 39 | jedis.close(); |

|  |  |
| --- | --- |
| 40 | } |

|  |  |
| --- | --- |
| 41 | } |

|  |  |
| --- | --- |
| 42 | } |

|  |  |
| --- | --- |
| 43 | LOG.info("Result[Server|Recv|RES] " + builder.toString()); |

|  |  |
| --- | --- |
| 44 | return builder.toString(); |

|  |  |
| --- | --- |
| 45 | } |

|  |  |
| --- | --- |
| 46 |  |

|  |  |
| --- | --- |
| 47 | @Override |

|  |  |
| --- | --- |
| 48 | public List<String> getMaxOnlineUserCount(List<String> rooms, String date, String dateFormat) { |

|  |  |
| --- | --- |
| 49 | // HGETALL chat::room::max::user::cnt::20150326 |

|  |  |
| --- | --- |
| 50 | LOG.info("Params[Server|Recv|REQ] rooms=" + rooms + ",date=" + date +",dateFormat=" + dateFormat); |

|  |  |
| --- | --- |
| 51 | String whichDate = DateTimeUtils.format(date, dateFormat, DF\_YYYYMMDD); |

|  |  |
| --- | --- |
| 52 | String key = KEY\_MAX\_USER\_COUNT\_PREFIX + whichDate; |

|  |  |
| --- | --- |
| 53 | StringBuffer builder = new StringBuffer(); |

|  |  |
| --- | --- |
| 54 | if(rooms != null && !rooms.isEmpty()) { |

|  |  |
| --- | --- |
| 55 | Jedis jedis = null; |

|  |  |
| --- | --- |
| 56 | try { |

|  |  |
| --- | --- |
| 57 | jedis = jedisPool.getResource(); |

|  |  |
| --- | --- |
| 58 | return jedis.hmget(key, rooms.toArray(new String[rooms.size()])); |

|  |  |
| --- | --- |
| 59 | } catch (Exception e) { |

|  |  |
| --- | --- |
| 60 | LOG.error("", e); |

|  |  |
| --- | --- |
| 61 | } finally { |

|  |  |
| --- | --- |
| 62 | if(jedis != null) { |

|  |  |
| --- | --- |
| 63 | jedis.close(); |

|  |  |
| --- | --- |
| 64 | } |

|  |  |
| --- | --- |
| 65 | } |

|  |  |
| --- | --- |
| 66 | } |

|  |  |
| --- | --- |
| 67 | LOG.info("Result[Server|Recv|RES] " + builder.toString()); |

|  |  |
| --- | --- |
| 68 | return Lists.newArrayList(); |

|  |  |
| --- | --- |
| 69 | } |

|  |  |
| --- | --- |
| 70 |  |

|  |  |
| --- | --- |
| 71 | public void setJedisPool(JedisPool jedisPool) { |

|  |  |
| --- | --- |
| 72 | this.jedisPool = jedisPool; |

|  |  |
| --- | --- |
| 73 | } |

|  |  |
| --- | --- |
| 74 |  |

|  |  |
| --- | --- |
| 75 | } |

代码中通过读取Redis中数据来完成调用，逻辑比较简单。对应的Maven POM依赖配置，如下所示：

|  |  |
| --- | --- |
| 01 | <dependencies> |

|  |  |
| --- | --- |
| 02 | <dependency> |

|  |  |
| --- | --- |
| 03 | <groupId>org.shirdrn.dubbo</groupId> |

|  |  |
| --- | --- |
| 04 | <artifactId>dubbo-api</artifactId> |

|  |  |
| --- | --- |
| 05 | <version>0.0.1-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 06 | </dependency> |

|  |  |
| --- | --- |
| 07 | <dependency> |

|  |  |
| --- | --- |
| 08 | <groupId>org.shirdrn.dubbo</groupId> |

|  |  |
| --- | --- |
| 09 | <artifactId>dubbo-commons</artifactId> |

|  |  |
| --- | --- |
| 10 | <version>0.0.1-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 11 | </dependency> |

|  |  |
| --- | --- |
| 12 | <dependency> |

|  |  |
| --- | --- |
| 13 | <groupId>redis.clients</groupId> |

|  |  |
| --- | --- |
| 14 | <artifactId>jedis</artifactId> |

|  |  |
| --- | --- |
| 15 | <version>2.5.2</version> |

|  |  |
| --- | --- |
| 16 | </dependency> |

|  |  |
| --- | --- |
| 17 | <dependency> |

|  |  |
| --- | --- |
| 18 | <groupId>org.apache.commons</groupId> |

|  |  |
| --- | --- |
| 19 | <artifactId>commons-pool2</artifactId> |

|  |  |
| --- | --- |
| 20 | <version>2.2</version> |

|  |  |
| --- | --- |
| 21 | </dependency> |

|  |  |
| --- | --- |
| 22 | <dependency> |

|  |  |
| --- | --- |
| 23 | <groupId>org.jboss.netty</groupId> |

|  |  |
| --- | --- |
| 24 | <artifactId>netty</artifactId> |

|  |  |
| --- | --- |
| 25 | <version>3.2.7.Final</version> |

|  |  |
| --- | --- |
| 26 | </dependency> |

|  |  |
| --- | --- |
| 27 | </dependencies> |

有关对Dubbo框架的一些依赖，我们单独放到一个通用的Maven Module中（详见后面“附录：Dubbo使用Maven构建依赖配置”），这里不再多说。服务提供方实现，最关键的就是服务的配置，因为Dubbo基于Spring来管理配置和实例，所以通过配置可以指定服务是否是分布式服务，以及通过配置增加很多其它特性。我们的配置文件为provider-cluster.xml，内容如下所示：

|  |  |
| --- | --- |
| 01 | <?xml version="1.0" encoding="UTF-8"?> |

|  |  |
| --- | --- |
| 02 |  |

|  |  |
| --- | --- |
| 03 | <beans xmlns="<http://www.springframework.org/schema/beans>" |

|  |  |
| --- | --- |
| 04 | [xmlns:xsi](http://xmlnsxsi)="<http://www.w3.org/2001/XMLSchema-instance>"[xmlns:dubbo](http://xmlnsdubbo)="<http://code.alibabatech.com/schema/dubbo>" |

|  |  |
| --- | --- |
| 05 | xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |
| --- | --- |
| 06 | [xsi:schemaLocation=](http://xsischemaLocation=)"<http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.0.xsd> |

|  |  |
| --- | --- |
| 07 | <http://code.alibabatech.com/schema/dubbo><http://code.alibabatech.com/schema/dubbo/dubbo.xsd>"> |

|  |  |
| --- | --- |
| 08 |  |

|  |  |
| --- | --- |
| 09 | <beanclass="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"> |

|  |  |
| --- | --- |
| 10 | <property name="systemPropertiesModeName"value="SYSTEM\_PROPERTIES\_MODE\_OVERRIDE" /> |

|  |  |
| --- | --- |
| 11 | <property name="ignoreResourceNotFound" value="true" /> |

|  |  |
| --- | --- |
| 12 | <property name="locations"> |

|  |  |
| --- | --- |
| 13 | <list> |

|  |  |
| --- | --- |
| 14 | <value>classpath\*:jedis.properties</value> |

|  |  |
| --- | --- |
| 15 | </list> |

|  |  |
| --- | --- |
| 16 | </property> |

|  |  |
| --- | --- |
| 17 | </bean> |

|  |  |
| --- | --- |
| 18 |  |

|  |  |
| --- | --- |
| 19 | <[dubbo:application](http://dubboapplication) name="chatroom-cluster-provider" /> |

|  |  |
| --- | --- |
| 20 | <[dubbo:registry](http://dubboregistry) address="<zookeeper://zk1:2181?backup=zk2:2181>,[zk3:2181](http://zk3:2181)" /> |

|  |  |
| --- | --- |
| 21 |  |

|  |  |
| --- | --- |
| 22 | <[dubbo:protocol](http://dubboprotocol) name="dubbo" port="20880" /> |

|  |  |
| --- | --- |
| 23 |  |

|  |  |
| --- | --- |
| 24 | <[dubbo:service](http://dubboservice)interface="org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService" version="1.0.0" |

|  |  |
| --- | --- |
| 25 | cluster="failover" retries="2" timeout="1000" loadbalance="random"actives="100" executes="200" |

|  |  |
| --- | --- |
| 26 | ref="chatRoomOnlineUserCounterService" protocol="dubbo" > |

|  |  |
| --- | --- |
| 27 | <[dubbo:method](http://dubbomethod) name="queryRoomUserCount" timeout="500" retries="2"loadbalance="roundrobin" actives="50" /> |

|  |  |
| --- | --- |
| 28 | </[dubbo:service](http://dubboservice)> |

|  |  |
| --- | --- |
| 29 |  |

|  |  |
| --- | --- |
| 30 | <bean id="chatRoomOnlineUserCounterService"class="org.shirdrn.dubbo.provider.service.ChatRoomOnlineUserCounterServiceImpl" > |

|  |  |
| --- | --- |
| 31 | <property name="jedisPool" ref="jedisPool" /> |

|  |  |
| --- | --- |
| 32 | </bean> |

|  |  |
| --- | --- |
| 33 |  |

|  |  |
| --- | --- |
| 34 | <bean id="jedisPool" class="redis.clients.jedis.JedisPool" destroy-method="destroy"> |

|  |  |
| --- | --- |
| 35 | <constructor-arg index="0"> |

|  |  |
| --- | --- |
| 36 | <bean class="org.apache.commons.pool2.impl.GenericObjectPoolConfig"> |

|  |  |
| --- | --- |
| 37 | <property name="maxTotal" value="${redis.pool.maxTotal}" /> |

|  |  |
| --- | --- |
| 38 | <property name="maxIdle" value="${redis.pool.maxIdle}" /> |

|  |  |
| --- | --- |
| 39 | <property name="minIdle" value="${redis.pool.minIdle}" /> |

|  |  |
| --- | --- |
| 40 | <property name="maxWaitMillis"value="${redis.pool.maxWaitMillis}" /> |

|  |  |
| --- | --- |
| 41 | <property name="testOnBorrow" value="${redis.pool.testOnBorrow}"/> |

|  |  |
| --- | --- |
| 42 | <property name="testOnReturn" value="${redis.pool.testOnReturn}"/> |

|  |  |
| --- | --- |
| 43 | <property name="testWhileIdle" value="true" /> |

|  |  |
| --- | --- |
| 44 | </bean> |

|  |  |
| --- | --- |
| 45 | </constructor-arg> |

|  |  |
| --- | --- |
| 46 | <constructor-arg index="1" value="${redis.host}" /> |

|  |  |
| --- | --- |
| 47 | <constructor-arg index="2" value="${redis.port}" /> |

|  |  |
| --- | --- |
| 48 | <constructor-arg index="3" value="${redis.timeout}" /> |

|  |  |
| --- | --- |
| 49 | </bean> |

|  |  |
| --- | --- |
| 50 |  |

|  |  |
| --- | --- |
| 51 | </beans> |

上面配置中，使用dubbo协议，集群容错模式为failover，服务级别负载均衡策略为random，方法级别负载均衡策略为roundrobin（它覆盖了服务级别的配置内容），其他一些配置内容可以参考Dubbo文档。我们这里是从Redis读取数据，所以使用了Redis连接池。  
启动服务示例代码如下所示：

|  |  |
| --- | --- |
| 01 | package org.shirdrn.dubbo.provider; |

|  |  |
| --- | --- |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import org.shirdrn.dubbo.provider.common.DubboServer; |

|  |  |
| --- | --- |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public class ChatRoomClusterServer { |

|  |  |
| --- | --- |
| 06 |  |

|  |  |
| --- | --- |
| 07 | public static void main(String[] args) throws Exception { |

|  |  |
| --- | --- |
| 08 | DubboServer.startServer("[classpath:provider-cluster.xml](http://classpathprovider-cluster.xml)"); |

|  |  |
| --- | --- |
| 09 | } |

|  |  |
| --- | --- |
| 10 |  |

|  |  |
| --- | --- |
| 11 | } |

上面调用了DubboServer类的静态方法startServer，如下所示：

|  |  |
| --- | --- |
| 01 | public static void startServer(String config) { |

|  |  |
| --- | --- |
| 02 | ClassPathXmlApplicationContext context = newClassPathXmlApplicationContext(config); |

|  |  |
| --- | --- |
| 03 | try { |

|  |  |
| --- | --- |
| 04 | context.start(); |

|  |  |
| --- | --- |
| 05 | System.in.read(); |

|  |  |
| --- | --- |
| 06 | } catch (IOException e) { |

|  |  |
| --- | --- |
| 07 | e.printStackTrace(); |

|  |  |
| --- | --- |
| 08 | } finally { |

|  |  |
| --- | --- |
| 09 | context.close(); |

|  |  |
| --- | --- |
| 10 | } |

|  |  |
| --- | --- |
| 11 | } |

方法中主要是初始化Spring IoC容器，全部对象都交由容器来管理。

* 服务消费方

服务消费方就容易了，只需要知道注册中心地址，并引用服务提供方提供的接口，消费方调用服务实现如下所示：

|  |  |
| --- | --- |
| 01 | package org.shirdrn.dubbo.consumer; |

|  |  |
| --- | --- |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.Arrays; |

|  |  |
| --- | --- |
| 04 | import java.util.List; |

|  |  |
| --- | --- |
| 05 |  |

|  |  |
| --- | --- |
| 06 | import org.apache.commons.logging.Log; |

|  |  |
| --- | --- |
| 07 | import org.apache.commons.logging.LogFactory; |

|  |  |
| --- | --- |
| 08 | import org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService; |

|  |  |
| --- | --- |
| 09 | import org.springframework.context.support.AbstractXmlApplicationContext; |

|  |  |
| --- | --- |
| 10 | import org.springframework.context.support.ClassPathXmlApplicationContext; |

|  |  |
| --- | --- |
| 11 |  |

|  |  |
| --- | --- |
| 12 | public class ChatRoomDubboConsumer { |

|  |  |
| --- | --- |
| 13 |  |

|  |  |
| --- | --- |
| 14 | private static final Log LOG = LogFactory.getLog(ChatRoomDubboConsumer.class); |

|  |  |
| --- | --- |
| 15 |  |

|  |  |
| --- | --- |
| 16 | public static void main(String[] args) throws Exception { |

|  |  |
| --- | --- |
| 17 | AbstractXmlApplicationContext context = newClassPathXmlApplicationContext("[classpath:consumer.xml](http://classpathconsumer.xml)"); |

|  |  |
| --- | --- |
| 18 | try { |

|  |  |
| --- | --- |
| 19 | context.start(); |

|  |  |
| --- | --- |
| 20 | ChatRoomOnlineUserCounterService chatRoomOnlineUserCounterService = (ChatRoomOnlineUserCounterService) context.getBean("chatRoomOnlineUserCounterService"); |

|  |  |
| --- | --- |
| 21 | getMaxOnlineUserCount(chatRoomOnlineUserCounterService); |

|  |  |
| --- | --- |
| 22 | getRealtimeOnlineUserCount(chatRoomOnlineUserCounterService); |

|  |  |
| --- | --- |
| 23 | System.in.read(); |

|  |  |
| --- | --- |
| 24 | } finally { |

|  |  |
| --- | --- |
| 25 | context.close(); |

|  |  |
| --- | --- |
| 26 | } |

|  |  |
| --- | --- |
| 27 |  |

|  |  |
| --- | --- |
| 28 | } |

|  |  |
| --- | --- |
| 29 |  |

|  |  |
| --- | --- |
| 30 | private static void getMaxOnlineUserCount(ChatRoomOnlineUserCounterService liveRoomOnlineUserCountService) { |

|  |  |
| --- | --- |
| 31 | List<String> maxUserCounts = liveRoomOnlineUserCountService.getMaxOnlineUserCount( |

|  |  |
| --- | --- |
| 32 | Arrays.asList(new String[] {"1482178010" , "1408492761","1430546839", "1412517075", "1435861734"}), "20150327", "yyyyMMdd"); |

|  |  |
| --- | --- |
| 33 | LOG.info("After getMaxOnlineUserCount invoked: maxUserCounts= " + maxUserCounts) |

|  |  |
| --- | --- |
| 34 | } |

|  |  |
| --- | --- |
| 35 |  |

|  |  |
| --- | --- |
| 36 | private static void getRealtimeOnlineUserCount(ChatRoomOnlineUserCounterService liveRoomOnlineUserCountService) |

|  |  |
| --- | --- |
| 37 | throws InterruptedException { |

|  |  |
| --- | --- |
| 38 | String rooms = "1482178010,1408492761,1430546839,1412517075,1435861734"; |

|  |  |
| --- | --- |
| 39 | String onlineUserCounts = liveRoomOnlineUserCountService.queryRoomUserCount(rooms); |

|  |  |
| --- | --- |
| 40 | LOG.info("After queryRoomUserCount invoked: onlineUserCounts= " + onlineUserCounts); |

|  |  |
| --- | --- |
| 41 | } |

|  |  |
| --- | --- |
| 42 | } |

对应的配置文件为consumer.xml，内容如下所示：

|  |  |
| --- | --- |
| 01 | <?xml version="1.0" encoding="UTF-8"?> |

|  |  |
| --- | --- |
| 02 |  |

|  |  |
| --- | --- |
| 03 | <beans xmlns="<http://www.springframework.org/schema/beans>" |

|  |  |
| --- | --- |
| 04 | [xmlns:xsi](http://xmlnsxsi)="<http://www.w3.org/2001/XMLSchema-instance>"[xmlns:dubbo](http://xmlnsdubbo)="<http://code.alibabatech.com/schema/dubbo>" |

|  |  |
| --- | --- |
| 05 | [xsi:schemaLocation=](http://xsischemaLocation=)"<http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.0.xsd> |

|  |  |
| --- | --- |
| 06 | <http://code.alibabatech.com/schema/dubbo><http://code.alibabatech.com/schema/dubbo/dubbo.xsd>"> |

|  |  |
| --- | --- |
| 07 |  |

|  |  |
| --- | --- |
| 08 | <[dubbo:application](http://dubboapplication) name="chatroom-consumer" /> |

|  |  |
| --- | --- |
| 09 | <[dubbo:registry](http://dubboregistry) address="<zookeeper://zk1:2181?backup=zk2:2181>,[zk3:2181](http://zk3:2181)" /> |

|  |  |
| --- | --- |
| 10 |  |

|  |  |
| --- | --- |
| 11 | <[dubbo:reference](http://dubboreference) id="chatRoomOnlineUserCounterService"interface="org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService" version="1.0.0"> |

|  |  |
| --- | --- |
| 12 | <[dubbo:method](http://dubbomethod) name="queryRoomUserCount" retries="2" /> |

|  |  |
| --- | --- |
| 13 | </[dubbo:reference](http://dubboreference)> |

|  |  |
| --- | --- |
| 14 |  |

|  |  |
| --- | --- |
| 15 | </beans> |

也可以根据需要配置dubbo:reference相关的属性值，也可以配置dubbo:method指定调用的方法的配置信息，详细配置属性可以参考Dubbo官方文档。

* 部署与验证

开发完成提供方服务后，在本地开发调试的时候可以怎么简单怎么做，如果是要部署到生产环境，则需要打包后进行部署，可以参考下面的Maven POM配置：

|  |  |
| --- | --- |
| 01 | <build> |

|  |  |
| --- | --- |
| 02 | <plugins> |

|  |  |
| --- | --- |
| 03 | <plugin> |

|  |  |
| --- | --- |
| 04 | <groupId>org.apache.maven.plugins</groupId> |

|  |  |
| --- | --- |
| 05 | <artifactId>maven-shade-plugin</artifactId> |

|  |  |
| --- | --- |
| 06 | <version>1.4</version> |

|  |  |
| --- | --- |
| 07 | <configuration> |

|  |  |
| --- | --- |
| 08 | <createDependencyReducedPom>true</createDependencyReducedPom> |

|  |  |
| --- | --- |
| 09 | </configuration> |

|  |  |
| --- | --- |
| 10 | <executions> |

|  |  |
| --- | --- |
| 11 | <execution> |

|  |  |
| --- | --- |
| 12 | <phase>package</phase> |

|  |  |
| --- | --- |
| 13 | <goals> |

|  |  |
| --- | --- |
| 14 | <goal>shade</goal> |

|  |  |
| --- | --- |
| 15 | </goals> |

|  |  |
| --- | --- |
| 16 | <configuration> |

|  |  |
| --- | --- |
| 17 | <transformers> |

|  |  |
| --- | --- |
| 18 | <transformerimplementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/> |

|  |  |
| --- | --- |
| 19 | <transformerimplementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"> |

|  |  |
| --- | --- |
| 20 | <mainClass>org.shirdrn.dubbo.provider.ChatRoomClusterServer</mainClass> |

|  |  |
| --- | --- |
| 21 | </transformer> |

|  |  |
| --- | --- |
| 22 | </transformers> |

|  |  |
| --- | --- |
| 23 | </configuration> |

|  |  |
| --- | --- |
| 24 | </execution> |

|  |  |
| --- | --- |
| 25 | </executions> |

|  |  |
| --- | --- |
| 26 | </plugin> |

|  |  |
| --- | --- |
| 27 | </plugins> |

|  |  |
| --- | --- |
| 28 | </build> |

这里也给出Maven POM依赖的简单配置：

|  |  |
| --- | --- |
| 1 | <dependencies> |

|  |  |
| --- | --- |
| 2 | <dependency> |

|  |  |
| --- | --- |
| 3 | <groupId>org.shirdrn.dubbo</groupId> |

|  |  |
| --- | --- |
| 4 | <artifactId>dubbo-api</artifactId> |

|  |  |
| --- | --- |
| 5 | <version>0.0.1-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 6 | </dependency> |

|  |  |
| --- | --- |
| 7 | </dependencies> |

我们开发的服务应该是分布式的，首先是通过配置内容来决定，例如设置集群模式、设置负载均衡模式等，然后在部署的时候，可以在多个节点上同一个服务，这样多个服务都会注册到Dubbo注册中心，如果某个节点上的服务不可用了，可以根据我们配置的策略来选择其他节点上的可用服务，后面通过Dubbo服务管理中心和监控中心就能更加清楚明了。

**Dubbo服务管理与监控**

我们需要在安装好管理中心和监控中心以后，再将上面的开发的提供方服务部署到物理节点上，然后就能够通过管理中心和监控中心来查看对应的详细情况。

* Dubbo服务管理中心

安装Dubbo服务管理中心，需要选择一个Web容器，我们使用Tomcat服务器。首先下载Dubbo管理中心安装文件dubbo-admin-2.5.3.war，或者直接从源码构建得到该WAR文件。这里，我们已经构建好对应的WAR文件，然后进行安装，执行如下命令：

|  |  |
| --- | --- |
| 1 | cd apache-tomcat-6.0.35 |

|  |  |
| --- | --- |
| 2 | rm -rf webapps/ROOT |

|  |  |
| --- | --- |
| 3 | unzip ~/dubbo-admin-2.5.3.war -d webapps/ROOT |

修改配置文件~/apache-tomcat-6.0.35/webapps/ROOT/WEB-INF/dubbo.properties，指定我们的注册中心地址以及登录密码，内容如下所示：

|  |  |
| --- | --- |
| 1 | dubbo.registry.address=<zookeeper://zk1:2181?backup=zk2:2181>,[zk3:2181](http://zk3:2181) |

|  |  |
| --- | --- |
| 2 | dubbo.admin.root.password=root |

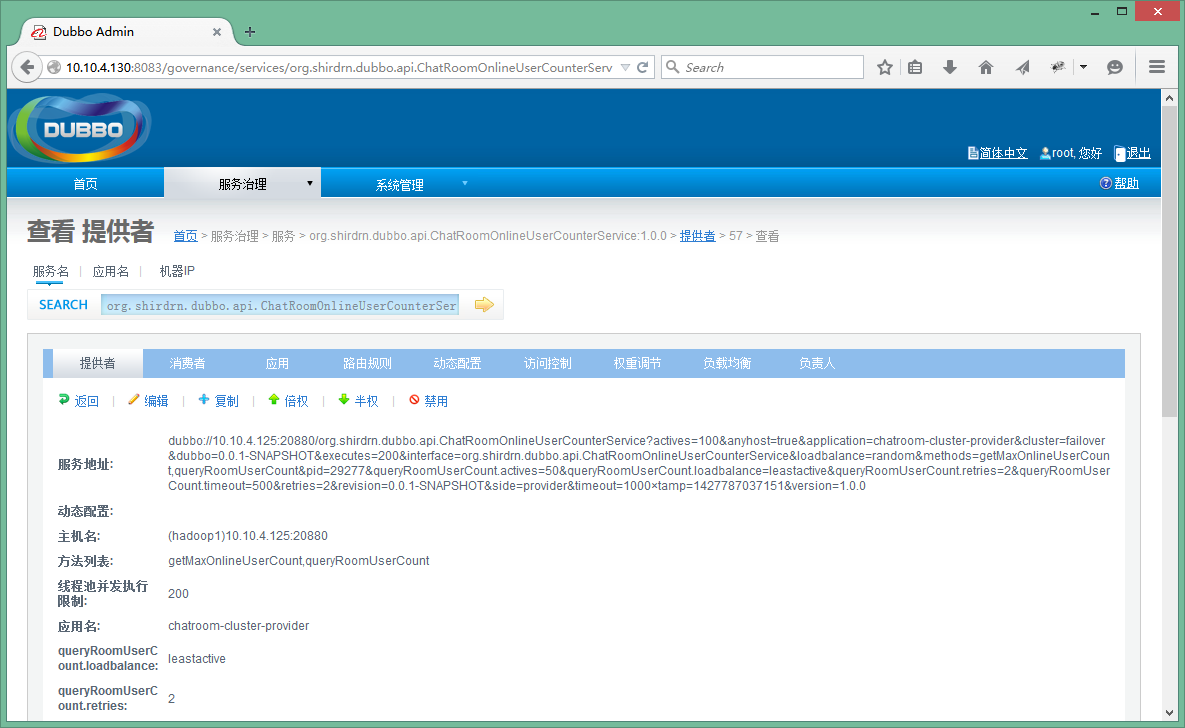
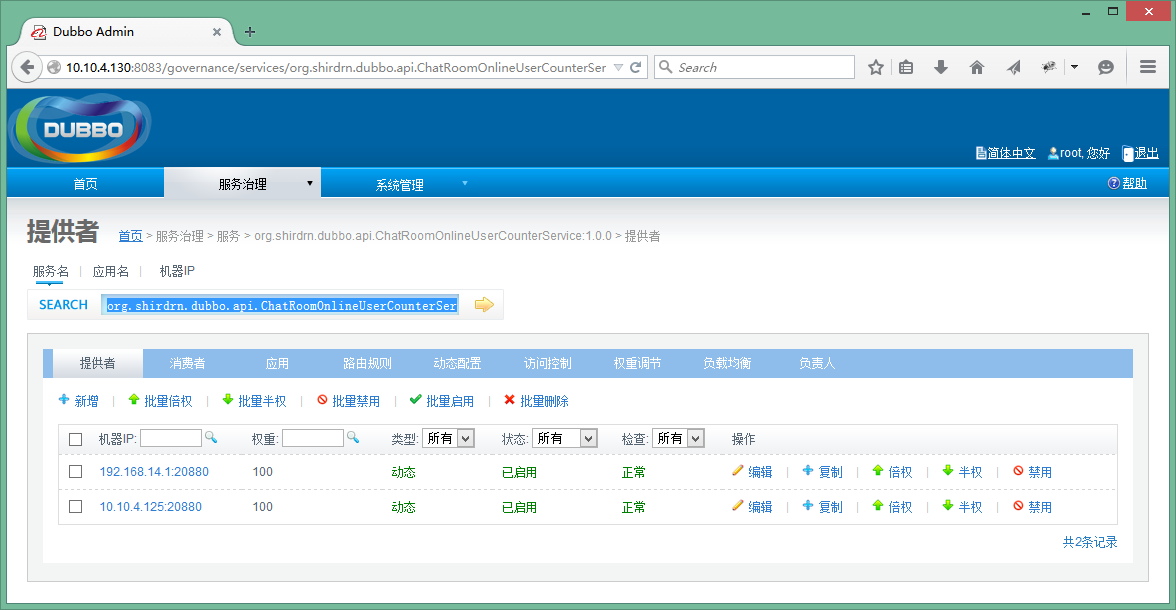
|  |  |
| --- | --- |
| 3 | dubbo.admin.guest.password=guest |

然后，根据需要修改~/apache-tomcat-6.0.35/conf/server.xml配置文件，主要是Tomcat HTTP 端口号（我这里使用8083端口），完成后可以直接启动Tomcat服务器：

|  |  |
| --- | --- |
| 1 | cd ~/apache-tomcat-6.0.35/ |

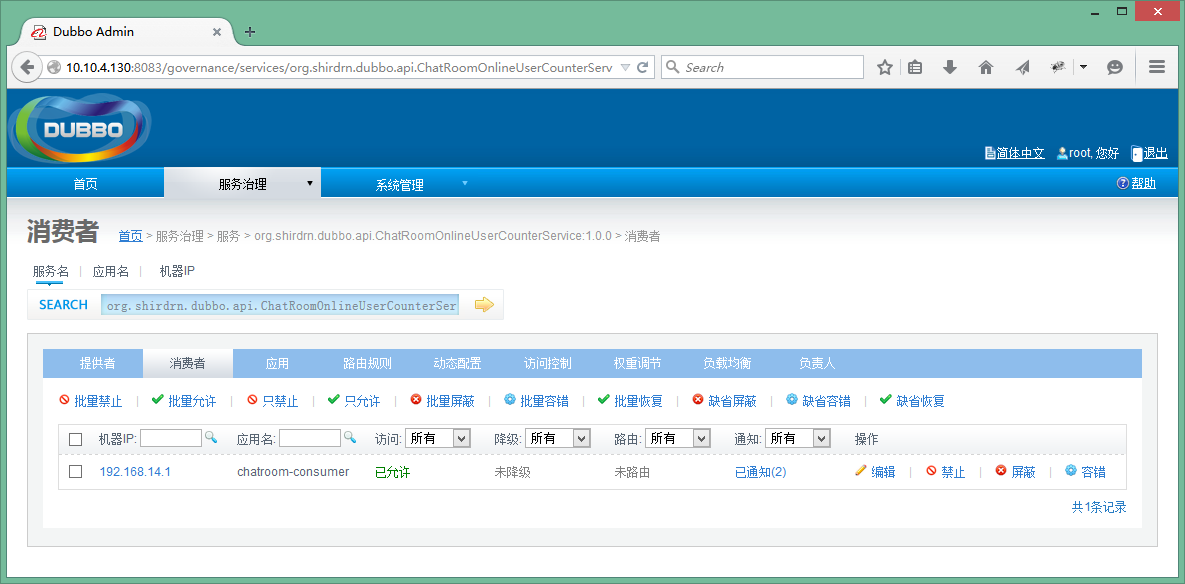
|  |  |
| --- | --- |
| 2 | bin/catalina.sh start |

然后访问地址<http://10.10.4.130:8083/>即可，根据配置文件指定的root用户密码，就可以登录Dubbo管理控制台。  
我们将上面开发的服务提供方服务，部署到2个独立的节点上（192.168.14.1和10.10.4.125），然后可以通过Dubbo管理中心查看对应服务的状况，如图所示：  
  
上图中可以看出，该服务有两个独立的节点可以提供，因为配置的集群模式为failover，如果某个节点的服务发生故障无法使用，则会自动透明地重试另一个节点上的服务，这样就不至于出现拒绝服务的情况。如果想要查看提供方某个节点上的服务详情，可以点击对应的IP:Port链接，示例如图所示：  
  
上图可以看到服务地址：



|  |  |
| --- | --- |
| 1 | <dubbo://10.10.4.125:20880/org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService?actives=100&anyhost=true&application=chatroom-cluster-provider&cluster=failover&dubbo=0.0.1-SNAPSHOT&executes=200&interface=org.shirdrn.dubbo.api.ChatRoomOnlineUserCounterService&loadbalance=random&methods=getMaxOnlineUserCount>,queryRoomUserCount&pid=30942&queryRoomUserCount.actives=50&queryRoomUserCount.loadbalance=leastactive&queryRoomUserCount.retries=2&queryRoomUserCount.timeout=500&retries=2&revision=0.0.1-SNAPSHOT&side=provider&timeout=1000×tamp=1427793652814&version=1.0.0 |

如果我们直接暴露该地址也是可以的，不过这种直连的方式对服务消费方不是透明的，如果以后IP地址更换，也会影响调用方，所以最好是通过注册中心来隐蔽服务地址。同一个服务所部署在的多个节点上，也就对应对应着多个服务地址。另外，也可以对已经发布的服务进行控制，如修改访问控制、负载均衡相关配置内容等，可以通过上图中“消费者”查看服务消费方调用服务的情况，如图所示：  
  
也在管理控制台可以对消费方进行管理控制。



* Dubbo监控中心

Dubbo监控中心是以Dubbo服务的形式发布到注册中心，和普通的服务时一样的。例如，我这里下载了Dubbo自带的简易监控中心文件dubbo-monitor-simple-2.5.3-assembly.tar.gz，可以解压缩以后，修改配置文件~/dubbo-monitor-simple-2.5.3/conf/dubbo.properties的内容，如下所示：

|  |  |
| --- | --- |
| 01 | dubbo.container=log4j,spring,registry,jetty |

|  |  |
| --- | --- |
| 02 | dubbo.application.name=simple-monitor |

|  |  |
| --- | --- |
| 03 | dubbo.application.owner= |

|  |  |
| --- | --- |
| 04 | dubbo.registry.address=<zookeeper://zk1:2181?backup=zk2:2181>,[zk3:2181](http://zk3:2181) |

|  |  |
| --- | --- |
| 05 | dubbo.protocol.port=7070 |

|  |  |
| --- | --- |
| 06 | dubbo.jetty.port=8087 |

|  |  |
| --- | --- |
| 07 | dubbo.jetty.directory=${user.home}/monitor |

|  |  |
| --- | --- |
| 08 | dubbo.charts.directory=${dubbo.jetty.directory}/charts |

|  |  |
| --- | --- |
| 09 | dubbo.statistics.directory=${user.home}/monitor/statistics |

|  |  |
| --- | --- |
| 10 | dubbo.log4j.file=logs/dubbo-monitor-simple.log |

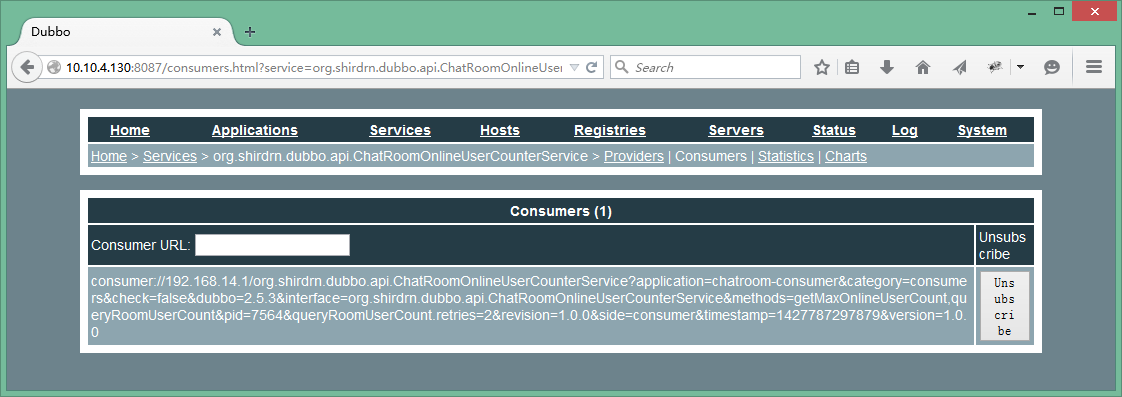
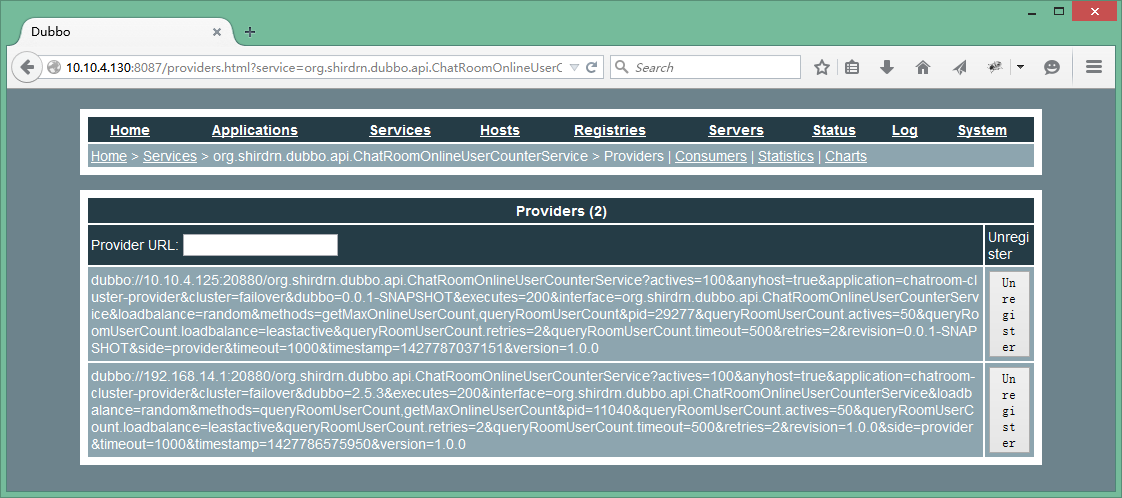
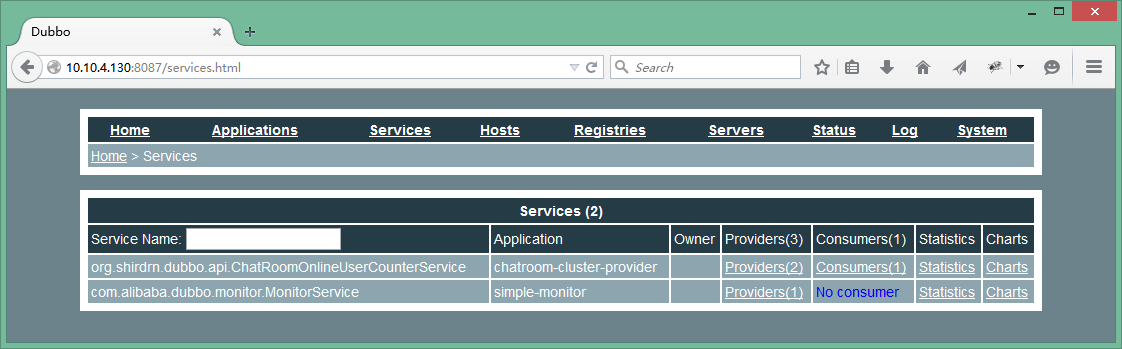
|  |  |
| --- | --- |
| 11 | dubbo.log4j.level=WARN |

然后启动简易监控中心，执行如下命令：

|  |  |
| --- | --- |
| 1 | cd ~/dubbo-monitor-simple-2.5.3 |

|  |  |
| --- | --- |
| 2 | bin/start.sh |

这里使用了Jetty Web容器，访问地址<http://10.10.4.130:8087/>就可以查看监控中心，Applications选项卡页面包含了服务提供方和消费方的基本信息，如图所示：  
  
上图主要列出了所有提供方发布的服务、消费方调用、服务依赖关系等内容。  
接着，查看Services选项卡页面，包含了服务提供方提供的服务列表，如图所示：  
  
点击上图中Providers链接就能看到服务提供方的基本信息，包括服务地址等，如图所示：  
  
点击上图中Consumers链接就能看到服务消费方的基本信息，包括服务地址等，如图所示：  
  
由于上面是Dubbo自带的一个简易监控中心，可能所展现的内容并不能满足我们的需要，所以可以根据需要开发自己的监控中心。Dubbo也提供了监控中心的扩展接口，如果想要实现自己的监控中心，可以实现接口com.alibaba.dubbo.monitor.MonitorFactory和com.alibaba.dubbo.monitor.Monitor，其中MonitorFactory接口定义如下所示：



|  |  |
| --- | --- |
| 01 | /\*\* |

|  |  |
| --- | --- |
| 02 | \* MonitorFactory. (SPI, Singleton, ThreadSafe) |

|  |  |
| --- | --- |
| 03 | \* |

|  |  |
| --- | --- |
| 04 | \* @author william.liangf |

|  |  |
| --- | --- |
| 05 | \*/ |

|  |  |
| --- | --- |
| 06 | @SPI("dubbo") |

|  |  |
| --- | --- |
| 07 | public interface MonitorFactory { |

|  |  |
| --- | --- |
| 08 |  |

|  |  |
| --- | --- |
| 09 | /\*\* |

|  |  |
| --- | --- |
| 10 | \* Create monitor. |

|  |  |
| --- | --- |
| 11 | \* @param url |

|  |  |
| --- | --- |
| 12 | \* @return monitor |

|  |  |
| --- | --- |
| 13 | \*/ |

|  |  |
| --- | --- |
| 14 | @Adaptive("protocol") |

|  |  |
| --- | --- |
| 15 | Monitor getMonitor(URL url); |

|  |  |
| --- | --- |
| 16 |  |

|  |  |
| --- | --- |
| 17 | } |

Monitor接口定义如下所示：

|  |  |
| --- | --- |
| 1 | /\*\* |

|  |  |
| --- | --- |
| 2 | \* Monitor. (SPI, Prototype, ThreadSafe) |

|  |  |
| --- | --- |
| 3 | \* |

|  |  |
| --- | --- |
| 4 | \* @see com.alibaba.dubbo.monitor.MonitorFactory#getMonitor(com.alibaba.dubbo.common.URL) |

|  |  |
| --- | --- |
| 5 | \* @author william.liangf |

|  |  |
| --- | --- |
| 6 | \*/ |

|  |  |
| --- | --- |
| 7 | public interface Monitor extends Node, MonitorService { |

|  |  |
| --- | --- |
| 8 |  |

|  |  |
| --- | --- |
| 9 | } |

具体定义内容可以查看MonitorService接口，不再累述。

**总结**

Dubbo还提供了其他很多高级特性，如路由规则、参数回调、服务分组、服务降级等等，而且很多特性在给出内置实现的基础上，还给出了扩展的接口，我们可以给出自定义的实现，非常方便而且强大。更多可以参考Dubbo官网用户手册和开发人员手册。

**附录：Dubbo使用Maven构建依赖配置**

|  |  |
| --- | --- |
| 01 | <properties> |

|  |  |
| --- | --- |
| 02 | <spring.version>3.2.8.RELEASE</spring.version> |

|  |  |
| --- | --- |
| 03 | <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding> |

|  |  |
| --- | --- |
| 04 | </properties> |

|  |  |
| --- | --- |
| 05 |  |

|  |  |
| --- | --- |
| 06 | <dependencies> |

|  |  |
| --- | --- |
| 07 | <dependency> |

|  |  |
| --- | --- |
| 08 | <groupId>com.alibaba</groupId> |

|  |  |
| --- | --- |
| 09 | <artifactId>dubbo</artifactId> |

|  |  |
| --- | --- |
| 10 | <version>2.5.3</version> |

|  |  |
| --- | --- |
| 11 | <exclusions> |

|  |  |
| --- | --- |
| 12 | <exclusion> |

|  |  |
| --- | --- |
| 13 | <groupId>org.springframework</groupId> |

|  |  |
| --- | --- |
| 14 | <artifactId>spring</artifactId> |

|  |  |
| --- | --- |
| 15 | </exclusion> |

|  |  |
| --- | --- |
| 16 | <exclusion> |

|  |  |
| --- | --- |
| 17 | <groupId>org.apache.zookeeper</groupId> |

|  |  |
| --- | --- |
| 18 | <artifactId>zookeeper</artifactId> |

|  |  |
| --- | --- |
| 19 | </exclusion> |

|  |  |
| --- | --- |
| 20 | <exclusion> |

|  |  |
| --- | --- |
| 21 | <groupId>org.jboss.netty</groupId> |

|  |  |
| --- | --- |
| 22 | <artifactId>netty</artifactId> |

|  |  |
| --- | --- |
| 23 | </exclusion> |

|  |  |
| --- | --- |
| 24 | </exclusions> |

|  |  |
| --- | --- |
| 25 | </dependency> |

|  |  |
| --- | --- |
| 26 | <dependency> |

|  |  |
| --- | --- |
| 27 | <groupId>org.springframework</groupId> |

|  |  |
| --- | --- |
| 28 | <artifactId>spring-core</artifactId> |

|  |  |
| --- | --- |
| 29 | <version>${spring.version}</version> |

|  |  |
| --- | --- |
| 30 | </dependency> |

|  |  |
| --- | --- |
| 31 | <dependency> |

|  |  |
| --- | --- |
| 32 | <groupId>org.springframework</groupId> |

|  |  |
| --- | --- |
| 33 | <artifactId>spring-beans</artifactId> |

|  |  |
| --- | --- |
| 34 | <version>${spring.version}</version> |

|  |  |
| --- | --- |
| 35 | </dependency> |

|  |  |
| --- | --- |
| 36 | <dependency> |

|  |  |
| --- | --- |
| 37 | <groupId>org.springframework</groupId> |

|  |  |
| --- | --- |
| 38 | <artifactId>spring-context</artifactId> |

|  |  |
| --- | --- |
| 39 | <version>${spring.version}</version> |

|  |  |
| --- | --- |
| 40 | </dependency> |

|  |  |
| --- | --- |
| 41 | <dependency> |

|  |  |
| --- | --- |
| 42 | <groupId>org.springframework</groupId> |

|  |  |
| --- | --- |
| 43 | <artifactId>spring-context-support</artifactId> |

|  |  |
| --- | --- |
| 44 | <version>${spring.version}</version> |

|  |  |
| --- | --- |
| 45 | </dependency> |

|  |  |
| --- | --- |
| 46 | <dependency> |

|  |  |
| --- | --- |
| 47 | <groupId>org.springframework</groupId> |

|  |  |
| --- | --- |
| 48 | <artifactId>spring-web</artifactId> |

|  |  |
| --- | --- |
| 49 | <version>${spring.version}</version> |

|  |  |
| --- | --- |
| 50 | </dependency> |

|  |  |
| --- | --- |
| 51 |  |

|  |  |
| --- | --- |
| 52 | <dependency> |

|  |  |
| --- | --- |
| 53 | <groupId>org.slf4j</groupId> |

|  |  |
| --- | --- |
| 54 | <artifactId>slf4j-api</artifactId> |

|  |  |
| --- | --- |
| 55 | <version>1.6.2</version> |

|  |  |
| --- | --- |
| 56 | </dependency> |

|  |  |
| --- | --- |
| 57 | <dependency> |

|  |  |
| --- | --- |
| 58 | <groupId>log4j</groupId> |

|  |  |
| --- | --- |
| 59 | <artifactId>log4j</artifactId> |

|  |  |
| --- | --- |
| 60 | <version>1.2.16</version> |

|  |  |
| --- | --- |
| 61 | </dependency> |

|  |  |
| --- | --- |
| 62 | <dependency> |

|  |  |
| --- | --- |
| 63 | <groupId>org.javassist</groupId> |

|  |  |
| --- | --- |
| 64 | <artifactId>javassist</artifactId> |

|  |  |
| --- | --- |
| 65 | <version>3.15.0-GA</version> |

|  |  |
| --- | --- |
| 66 | </dependency> |

|  |  |
| --- | --- |
| 67 | <dependency> |

|  |  |
| --- | --- |
| 68 | <groupId>com.alibaba</groupId> |

|  |  |
| --- | --- |
| 69 | <artifactId>hessian-lite</artifactId> |

|  |  |
| --- | --- |
| 70 | <version>3.2.1-fixed-2</version> |

|  |  |
| --- | --- |
| 71 | </dependency> |

|  |  |
| --- | --- |
| 72 | <dependency> |

|  |  |
| --- | --- |
| 73 | <groupId>com.alibaba</groupId> |

|  |  |
| --- | --- |
| 74 | <artifactId>fastjson</artifactId> |

|  |  |
| --- | --- |
| 75 | <version>1.1.8</version> |

|  |  |
| --- | --- |
| 76 | </dependency> |

|  |  |
| --- | --- |
| 77 | <dependency> |

|  |  |
| --- | --- |
| 78 | <groupId>org.jvnet.sorcerer</groupId> |

|  |  |
| --- | --- |
| 79 | <artifactId>sorcerer-javac</artifactId> |

|  |  |
| --- | --- |
| 80 | <version>0.8</version> |

|  |  |
| --- | --- |
| 81 | </dependency> |

|  |  |
| --- | --- |
| 82 | <dependency> |

|  |  |
| --- | --- |
| 83 | <groupId>org.apache.zookeeper</groupId> |

|  |  |
| --- | --- |
| 84 | <artifactId>zookeeper</artifactId> |

|  |  |
| --- | --- |
| 85 | <version>3.4.5</version> |

|  |  |
| --- | --- |
| 86 | </dependency> |

|  |  |
| --- | --- |
| 87 | <dependency> |

|  |  |
| --- | --- |
| 88 | <groupId>com.github.sgroschupf</groupId> |

|  |  |
| --- | --- |
| 89 | <artifactId>zkclient</artifactId> |

|  |  |
| --- | --- |
| 90 | <version>0.1</version> |

|  |  |
| --- | --- |
| 91 | </dependency> |

|  |  |
| --- | --- |
| 92 | <dependency> |

|  |  |
| --- | --- |
| 93 | <groupId>org.jboss.netty</groupId> |

|  |  |
| --- | --- |
| 94 | <artifactId>netty</artifactId> |

|  |  |
| --- | --- |
| 95 | <version>3.2.7.Final</version> |

|  |  |
| --- | --- |
| 96 | </dependency> |

|  |  |
| --- | --- |
| 97 | </dependencies> |