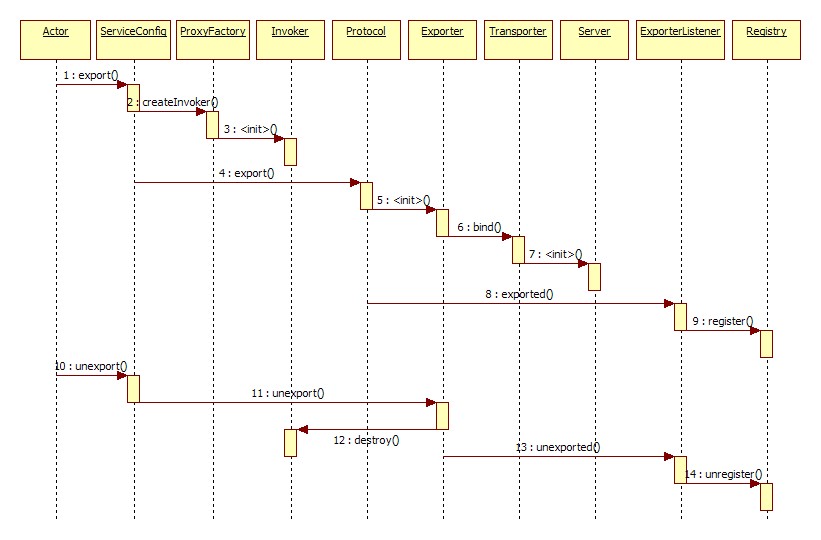
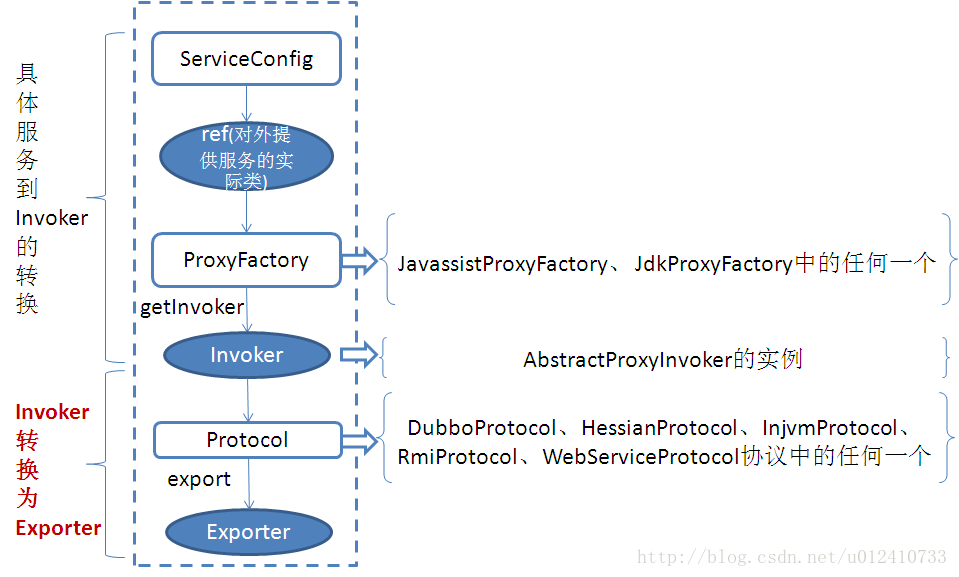
# Dubbo学习

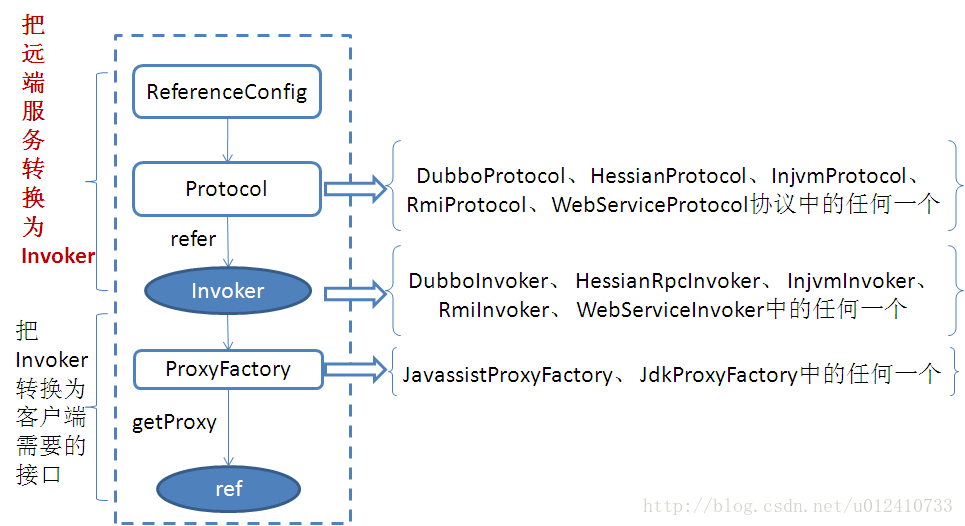
# 服务暴露：

<https://blog.csdn.net/u012410733/article/details/77417497>





# 服务调用：



# 集群容错

Dubbo提供的集群容错模式

**Failover Cluster：失败重试**

当服务消费方调用服务提供者失败后自动切换到其他服务提供者服务器进行重试。这通常用于读操作或者具有幂等的写操作，需要注意的是重试会带来更长延迟。可通过 retries="2" 来设置重试次数（不含第一次）。

接口级别配置重试次数方法 <dubbo:reference retries="2" /> ，如上配置当服务消费方调用服务失败后，会再重试两次，也就是说最多会做三次调用，这里的配置对该接口的所有方法生效。

**Failfast Cluster：快速失败**

当服务消费方调用服务提供者失败后，立即报错，也就是只调用一次。通常这种模式用于非幂等性的写操作。

**Failsafe Cluster：失败安全**

当服务消费者调用服务出现异常时，直接忽略异常。这种模式通常用于写入审计日志等操作。

**Failback Cluster：失败自动恢复**

当服务消费端用服务出现异常后，在后台记录失败的请求，并按照一定的策略后期再进行重试。这种模式通常用于消息通知操作。

**Forking Cluster：并行调用**

当消费方调用一个接口方法后，Dubbo Client会并行调用多个服务提供者的服务，只要一个成功即返回。这种模式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。可通过 forks="2" 来设置最大并行数。

**Broadcast Cluster：广播调用**

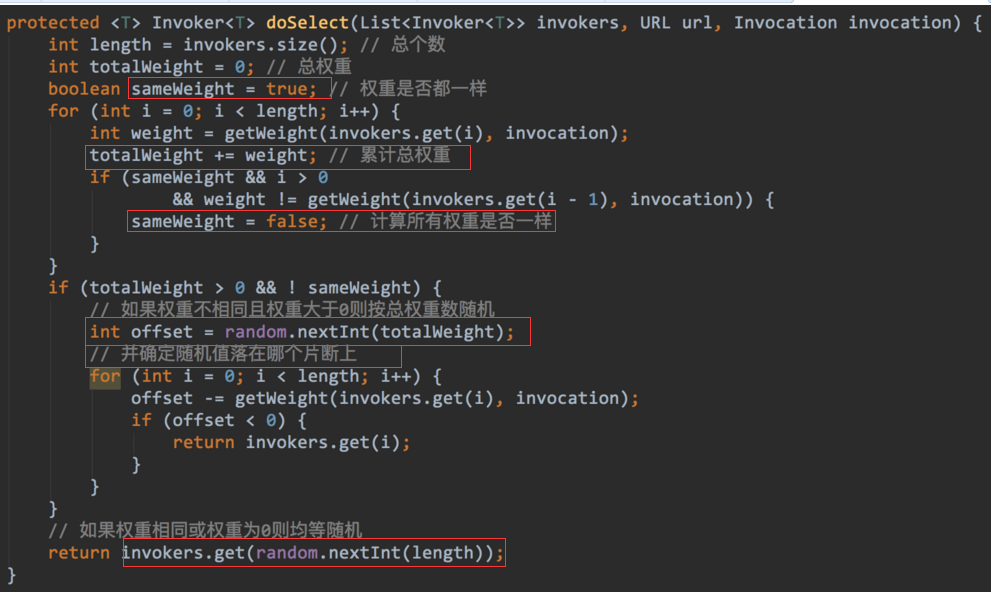
当消费者调用一个接口方法后，Dubbo Client会逐个调用所有服务提供者，任意一台调用异常则这次调用就标志失败。这种模式通常用于通知所有提供者更新缓存或日志等本地资源信息。

# 负载均衡

Dubbo默认为随机策略，另外提供轮询；最小活跃数，一致性hash策略。

## 随机策略

随机策略也有权重设置的，假设有四个集群节点A,B,C,D,对应的权重分别是1,2,3,4,那么请求到A节点的概率就为1/(1+2+3+4) = 10%.B,C,D节点依次类推为20%,30%,40%。



## 轮询策略

按照权重比例设置轮序比率。

缺点：若某一节点处理速度很慢，那么多次轮询后造成请求累计，最终导致宕机。

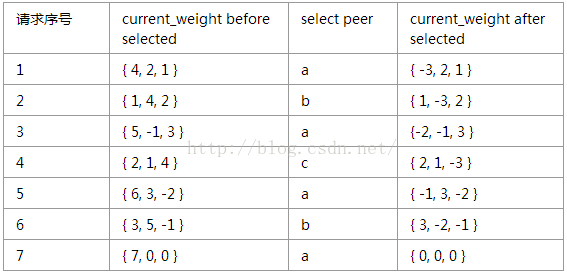
权重轮询算法设计：

注：Dubbo使用权重取模轮询。

最简单的，把服务器节点序列放在大小为总权重的数组中，如a权重为4，b为2，c为1，那么数组为{a,a,a,a,b,b,c}

但是上面的分布不均匀，所以用平滑算法来使服务器节点在数组中分布均匀。

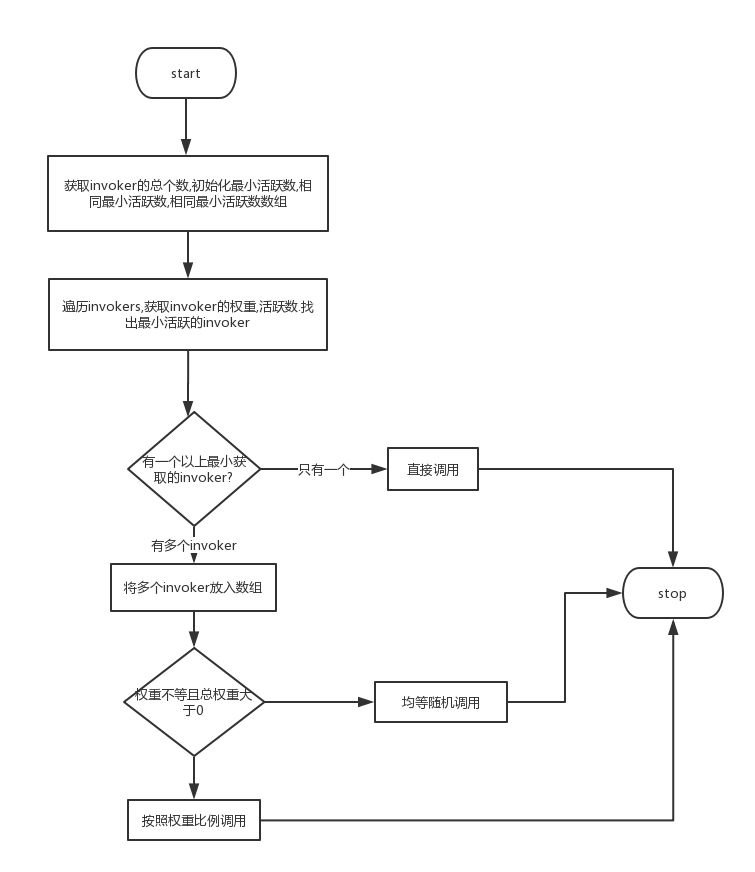
平滑算法：每个服务器节点有一个current\_weight属性，默认值为0；当第一个请求到达时，每个服务器的current\_weight都加上她自己的权重，然后本次请求分发给current\_weight最大的一台服务器，然后每个current\_weight减去总的权重，保存该数值。当下一个请求来临时，重复上述步骤。

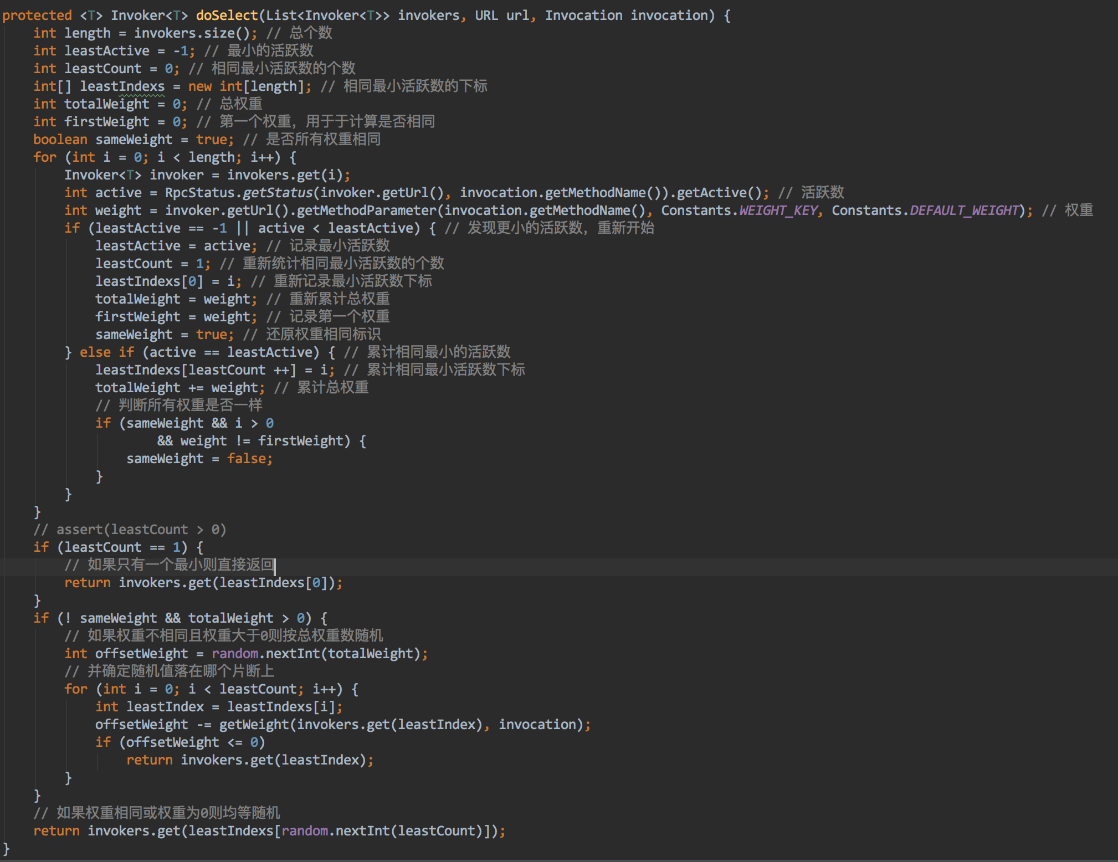


## 最小活跃数

最小活跃数算法可以实现使处理慢的服务器接收更少的请求。实现思路为：为每个服务器设置一个一个值，称为活跃度，记为active，默认值为0；当有一个请求到来时，active加一，处理完请求后，active减一。新的请求到来时，会选择活跃度小的服务器处理请求。

这部分代码概括起来就两部分,一部分是活跃数和权重的统计,另一部分是选择invoker.也就是他把最小活跃数的invoker统计到leastIndexs数组中,如果权重一致(这个一致的规则参考上面的随机算法)或者总权重为0,则均等随机调用,如果不同,则从leastIndexs数组中按照权重比例调用(还是和随机算法中的那个依次相减的思路一样).





## 一致性hash

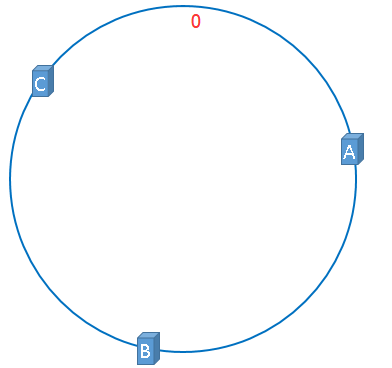
相同参数的请求总是发送到相同的服务器。当服务器挂时，该服务器的请求，基于虚拟节点，平摊到其他服务器，不会引起剧烈的波动。

一致性hash算法详解：

<http://www.zsythink.net/archives/1182>

<https://www.cnblogs.com/color-my-life/p/5799903.html>

我们对每个服务器对2^32次方取模，得到的是一个小于2^32次方的非负数，那么按照取模得到的数值将服务器放在一个圆上，这个圆有2^32个节点，如下图。



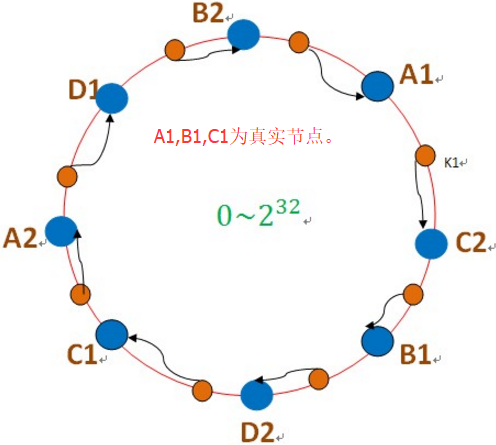
当需要存储数据时，对key进行2^32次方取模，假设得到的数值刚好等于某个服务器的位置值，则直接放进该服务器，如果不等，则顺时针寻找服务器，将数据放进找到的第一台服务器中。获取数据时也是同理。

当某个服务器挂了，则该服务器的数据会保存到它顺时针旋转命中的第一台服务器。

当新增一个节点，则将该节点所在位置值之前到它逆时针的第一台服务器之前的数据移动到该节点。

但是，上述方法是有缺点的。当A宕机时，A的数据迁移到B，若A的数据访问量很大，B的也不小，那么B必挂，同样迁移到C，C也必挂，引起了**雪崩。而且，A，B，C服务器也不一定会均匀分布。**

鉴于上述问题，引入虚拟节点。即可以在圆上，添加对于A，B，C的虚拟节点，使服务器分布均匀。如下图所示。



当A挂了，数据是分存储到C和D。避免了雪崩。

# 动态代理与RPC原理

<http://www.importnew.com/29359.html>

动态代理：核心是将分散的方法的调用统一到一个方法中来实现。比如：传奇霸业中的监听器处理，协议包处理。

# 线程模型

# 4、个人感受

个人觉得dubbo最主要的是动态代理与网络传输，线程模型，网络传输由协议和数据传输格式构成，dubbo支持：dubbo协议，webservice协议（rest+json风格与http+xml（soap），http协议等等。。。。。。

# 5、dubbo协议

<https://www.baidu.com/link?url=57aywD0Q6WTnl7XKbIHuE7lcWGXh50Vy3z1lItKlmdBJDMh15QrrdpFbWuxffLM1hn8M2dypPqFxPm8A0olv8UwmsqpE_h1IY0gQtLXnNXy&wd=&eqid=dac084ef00059fa0000000065b65aaad>

# 6、dubbo网络通信之netty

由netty线程模型牵引出dubbo线程模型。其中请求的有序处理实现跟霸业的MessageHandker（）类实现差不多。

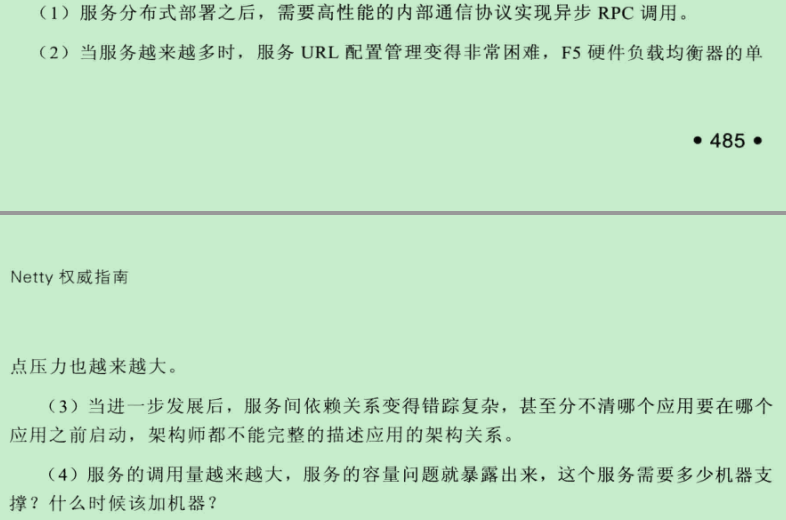
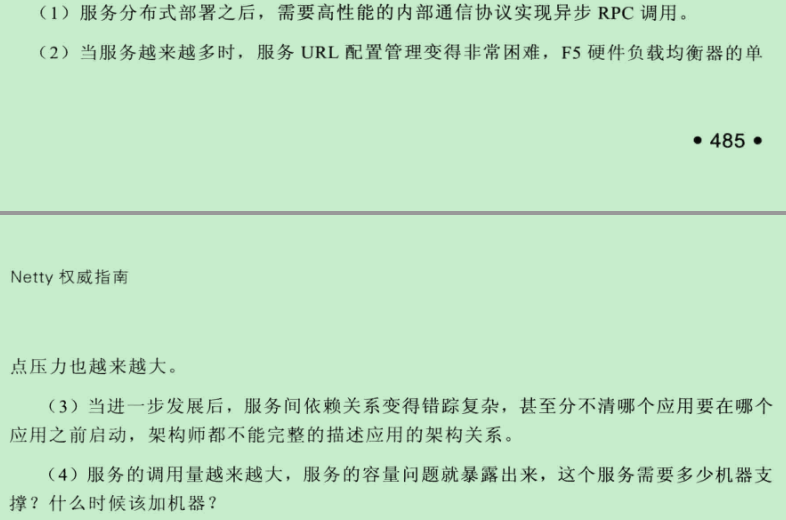
为啥需要有序处理：若使用线程池同时处理请求，假如一个玩家同时发了许多相同的请求过来，线程池中的所有线程均获得处理任务时，当这些请求是需要锁住玩家对象时，那么线程池中的其余线程就无法工作，造成浪费，所以需要请求的有序处理。

Dubbo的线程模型：<http://www.cnblogs.com/xhj123/p/9095278.html>

<https://blog.csdn.net/CHS007chs/article/details/73469480>

# 7、项目为什么用dubbo

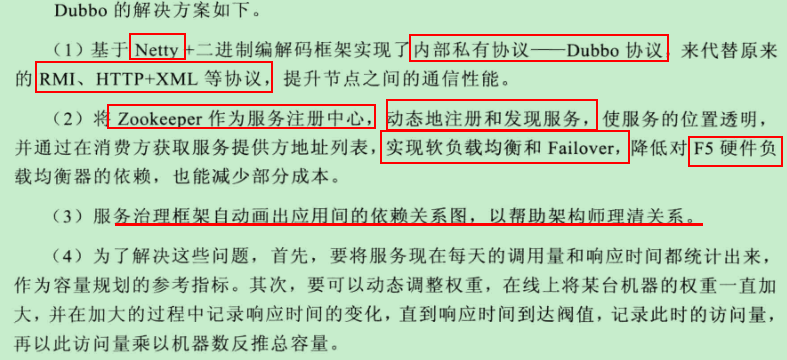
**SOA架构的服务之间通信存在的问题：**

如何实现远程通信？

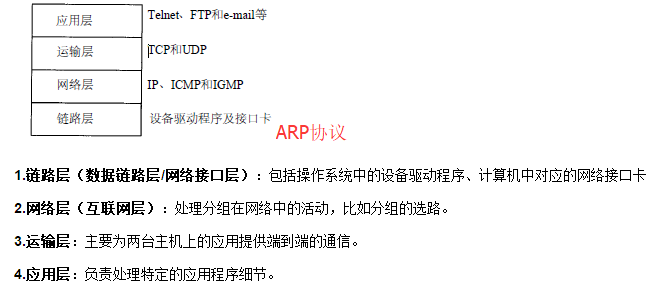
1. 使用WebService：效率不高,它是基于soap协议（http+xml）。接口描述会生成wsdl文件，消费者可以通过wsdl生成接口，导入到自己的工程中，然后通过soap协议实现调用。但项目中不推荐使用。
2. 使用restful形式的服务：http+json。使用httpclient模拟浏览器发送请求获得服务调用结果，很多项目中应用。如果服务越来越多，服务与服务之间的调用关系复杂，调用服务的URL管理复杂，什么时候添加机器难以确定。

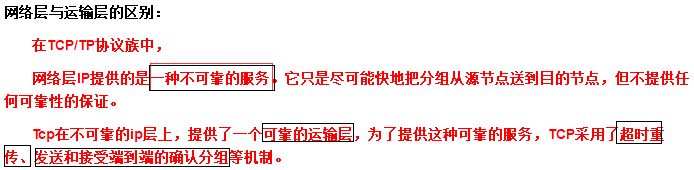
1. 使用dubbo。使用dubbo协议进行远程调用，直接使用socket通信。传输效率高，并且可以统计出系统之间的调用关系、调用次数，管理服务。



**具体需要看书：Netty权威指南第22章**

# 8、TCP/IP四层模型





Tcp滑动窗口：<https://www.cnblogs.com/woaiyy/p/3554182.html>

超时重传：<https://blog.csdn.net/qq_26499321/article/details/71429813>

# 9、dubbo串讲

<https://blog.csdn.net/wuzhengfei1112/article/details/77142147>