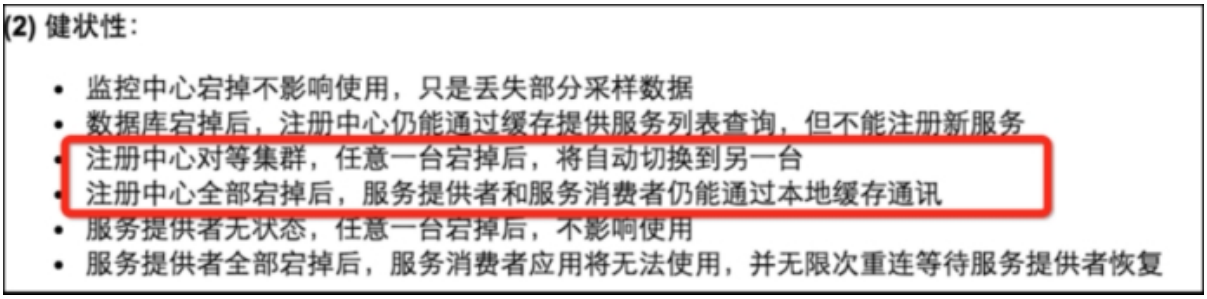
1. **Dubbo 中 zookeeper 做注册中心，如果注册中心集群都挂掉，发布者和订阅者之间还能通信么？**

可以通信的，启动 dubbo 时，消费者会从 zk 拉取注册的生产者的地址接口等数据，缓存在本地。每次调用时，按照本地存储的地址进行调用；

注册中心对等集群，任意一台宕机后，将会切换到另一台；注册中心全部宕机后，服务的提供者和消费者仍能通过本地缓存通讯。服务提供者无状态，任一台 宕机后，不影响使用；服务提供者全部宕机，服务消费者会无法使用，并无限次重连等待服务者恢复；

挂掉是不要紧的，但前提是你没有增加新的服务，如果你要调用新的服务，则是不能办到的。

附文档截图：



2. **dubbo 服务负载均衡策略？**

l Random LoadBalance

随机，按权重设置随机概率。在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。(权重可以在 dubbo 管控台配置)

l RoundRobin LoadBalance

轮循，按公约后的权重设置轮循比率。存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

l LeastActive LoadBalance

最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

l ConsistentHash LoadBalance

一致性 Hash，相同参数的请求总是发到同一提供者。当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。缺省只对第一个参数 Hash，如果要修改，请配置

[AppleScript] *纯文本查看 复制代码*

?

```
1    <dubbo:parameter key="hash.arguments" value="0,1" />
```

缺省用 160 份虚拟节点，如果要修改，请配置

[AppleScript] *纯文本查看 复制代码*

?

```
1    <dubbo:parameter key="hash.nodes" value="320" />
```

3. **Dubbo 在安全机制方面是如何解决的**

Dubbo 通过 Token 令牌防止用户绕过注册中心直连，然后在注册中心上管理授权。Dubbo 还提供服务黑白名单，来控制服务所允许的调用方。

4. dubbo 连接注册中心和直连的区别

在开发及测试环境下，经常需要绕过注册中心，只测试指定服务提供者，这时候可能需要点对点直连，

点对点直联方式，将以服务接口为单位，忽略注册中心的提供者列表，

服务注册中心，动态的注册和发现服务，使服务的位置透明，并通过在消费方获取服务提供方地址列表，实现软负载均衡和 Failover， 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。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，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，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。

1. dubbo 服务集群配置（集群容错模式）

在集群调用失败时，Dubbo 提供了多种容错方案，缺省为 failover 重试。可以自行扩展集群容错策略

l Failover Cluster(默认)

失败自动切换，当出现失败，重试其它服务器。(缺省)通常用于读操作，但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)。

[AppleScript] *纯文本查看* *复制代码*

?

```
<dubbo:service retries="2" cluster="failover"/>
      或：
      <dubbo:reference retries="2" cluster="failover"/>
      cluster="failover"可以不用写,因为默认就是 failover
```

l Failfast Cluster

快速失败，只发起一次调用，失败立即报错。通常用于非幂等性的写操作，比如新增记录。

[AppleScript] *纯文本查看* *复制代码*

?

```
dubbo:service cluster="failfast" />
      或：
      <dubbo:reference cluster="failfast" />
   cluster="failfast"和 把 cluster="failover"、retries="0"是一样的效果,retries="0"就是不重试
```

l Failsafe Cluster

失败安全，出现异常时，直接忽略。通常用于写入审计日志等操作。

[AppleScript] *纯文本查看* *复制代码*

?

```
<dubbo:service cluster="failsafe" />
      或：
      <dubbo:reference cluster="failsafe" />
```

l Failback Cluster

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。

[AppleScript] *纯文本查看 复制代码*

?

```
<dubbo:service cluster="failback" />
      或：
      <dubbo:reference cluster="failback" />
```

l Forking Cluster

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。可通过 forks="2"来设置最大并行数。

[AppleScript] *纯文本查看 复制代码*

?

```
<dubbo:service cluster="forking" forks="2"/>
      或：
      <dubbo:reference cluster="forking" forks="2"/>
```

l 配置

[AppleScript] *纯文本查看 复制代码*

?

```
服务端服务级别
          <dubbo:service interface="..." loadbalance="roundrobin" />
     客户端服务级别
          <dubbo:reference interface="..." loadbalance="roundrobin" />
     服务端方法级别          <dubbo:service interface="..."> <dubbo:method name="..." loadbalance
客户端方法级别          <dubbo:reference interface="..."> <dubbo:method name="..." loadbalance="
```

1. **dubbo 通信协议 dubbo 协议为什么要消费者比提供者个数多：**

因 dubbo 协议采用单一长连接，假设网络为千兆网卡(1024Mbit=128MByte)，

根据测试经验数据每条连接最多只能压满 7MByte(不同的环境可能不一样，供参考)，理论上 1 个服务提供者需要 20 个服务消费者才能压满网卡。

2. **dubbo 通信协议 dubbo 协议为什么不能传大包：**

因 dubbo 协议采用单一长连接，

如果每次请求的数据包大小为 500KByte，假设网络为千兆网卡(1024Mbit=128MByte)，每条连接最大 7MByte(不同的环境可能不一样，供参考)，

单个服务提供者的 TPS(每秒处理事务数)最大为：128MByte / 500KByte = 262。

单个消费者调用单个服务提供者的 TPS(每秒处理事务数)最大为：7MByte / 500KByte = 14。

如果能接受，可以考虑使用，否则网络将成为瓶颈。

3. **dubbo 通信协议 dubbo 协议为什么采用异步单一长连接：**

因为服务的现状大都是服务提供者少，通常只有几台机器，

而服务的消费者多，可能整个网站都在访问该服务，

比如 Morgan 的提供者只有 6 台提供者，却有上百台消费者，每天有 1.5 亿次调用，

如果采用常规的 hessian 服务，服务提供者很容易就被压跨，

通过单一连接，保证单一消费者不会压死提供者，

长连接，减少连接握手验证等，

并使用异步 IO，复用线程池，防止 C10K 问题。

4. **dubbo 通信协议 dubbo 协议适用范围和适用场景**

适用范围：传入传出参数数据包较小（建议小于 100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用 dubbo 协议传输大文件或超大字符串。

适用场景：常规远程服务方法调用

dubbo 协议补充：

连接个数：单连接

连接方式：长连接

传输协议：TCP

传输方式：NIO 异步传输

序列化：Hessian 二进制序列化

5. **RMI 协议**

RMI 协议采用 JDK 标准的 java.rmi.*实现，采用阻塞式短连接和 JDK 标准序列化方式，Java 标准的远程调用协议。

连接个数：多连接

连接方式：短连接

传输协议：TCP

传输方式：同步传输

序列化：Java 标准二进制序列化

适用范围：传入传出参数数据包大小混合，消费者与提供者个数差不多，可传文件。

适用场景：常规远程服务方法调用，与原生 RMI 服务互操作

6. **Hessian 协议**

Hessian 协议用于集成 Hessian 的服务，Hessian 底层采用 Http 通讯，采用 Servlet 暴露服务，Dubbo 缺省内嵌 Jetty 作为服务器实现

基于 Hessian 的远程调用协议。

连接个数：多连接

连接方式：短连接

传输协议：HTTP

传输方式：同步传输

序列化：Hessian 二进制序列化

适用范围：传入传出参数数据包较大，提供者比消费者个数多，提供者压力较大，可传文件。
适用场景：页面传输，文件传输，或与原生 hessian 服务互操作

7. **http**

采用 Spring 的 HttpInvoker 实现
基于 http 表单的远程调用协议。
连接个数：多连接
连接方式：短连接
传输协议：HTTP
传输方式：同步传输
序列化：表单序列化（JSON）
适用范围：传入传出参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或 URL 传入参数，暂不支持传文件。
适用场景：需同时给应用程序和浏览器 JS 使用的服务。

8. **Webservice**

基于 CXF 的 frontend-simple 和 transports-http 实现
基于 WebService 的远程调用协议。
连接个数：多连接
连接方式：短连接
传输协议：HTTP
传输方式：同步传输
序列化：SOAP 文本序列化
适用场景：系统集成，跨语言调用。

9. **Thrif**

Thrift 是 Facebook 捐给 Apache 的一个 RPC 框架，当前 dubbo 支持的 thrift 协议是对 thrift 原生协议的扩展，在原生协议的基础上添加了一些额外的头信息，比如 service name，magic number 等