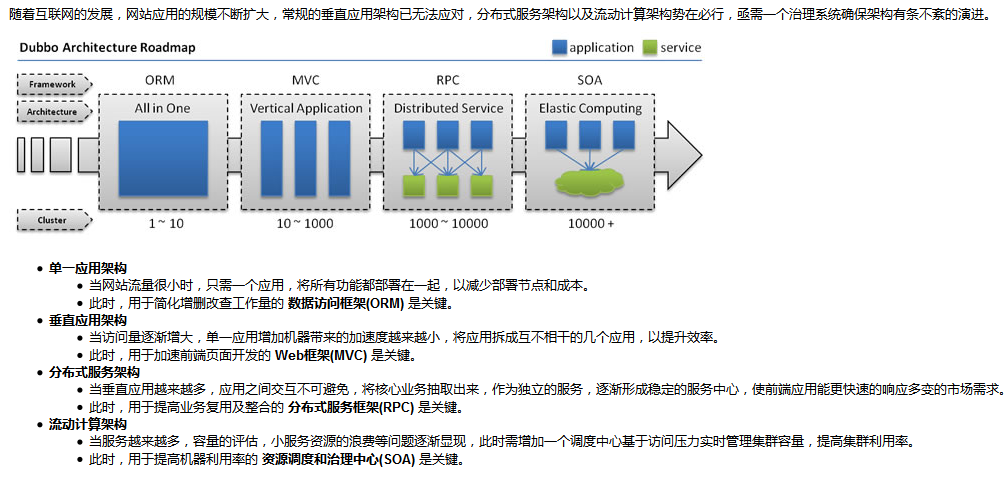
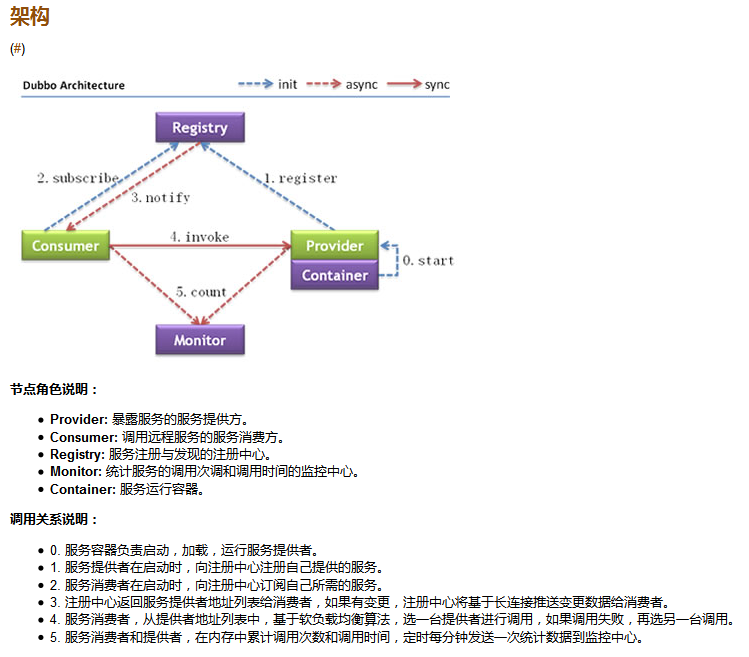
# 背景



# 2. 架构



说明:

**(1) 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

**(2) 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

**(3) 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

**(4) 升级性：**

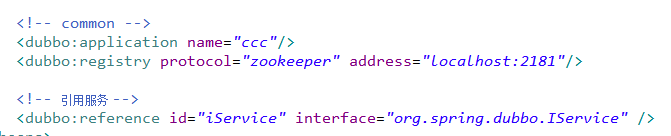
* 当服务集群规模进一步扩大，带动IT治理结构进一步升级，需要实现动态部署，进行流动计算，现有分布式服务架构不会带来阻力：

# 3. Demo

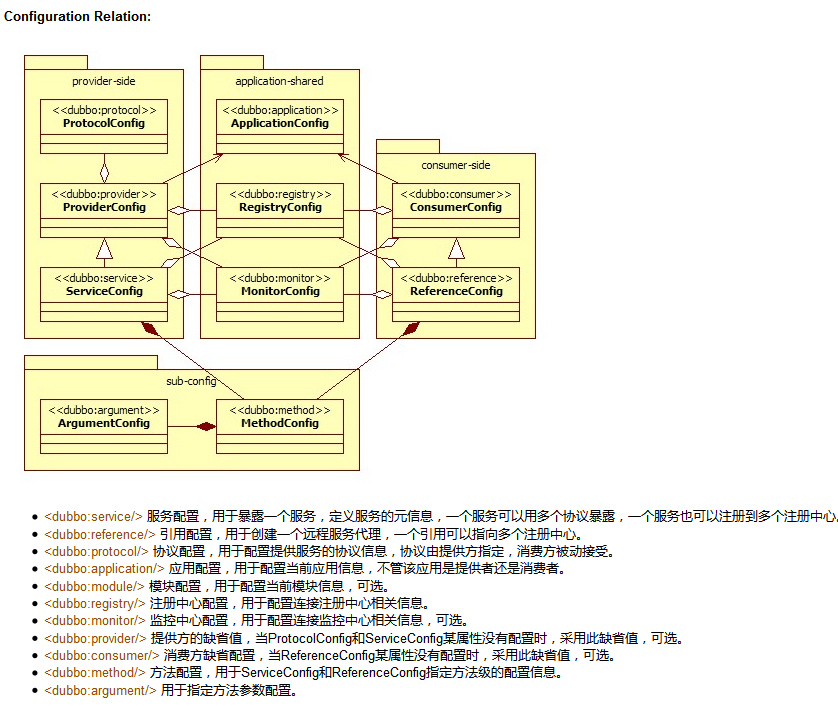
#服务提供方



#服务消费方



# 4. 配置类关系



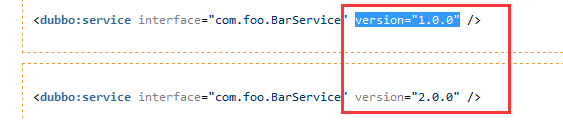
关于配置的说明

* 配置的查找顺序
  + 方法级优先，接口级次之，全局配置再次之。
  + 如果级别一样，则消费方优先，提供方次之。
* 其中，服务提供方配置，通过URL经由注册中心传递给消费方。
* 建议由服务提供方设置超时，因为一个方法需要执行多长时间，服务提供方更清楚，如果一个消费方同时引用多个服务，就不需要关心每个服务的超时设置。
* 理论上ReferenceConfig的非服务标识配置，在ConsumerConfig，ServiceConfig, ProviderConfig均可以缺省配置。

个人建议xml配置，更清晰和集中管理，较注解这种配置来说.

# 5. 知识点

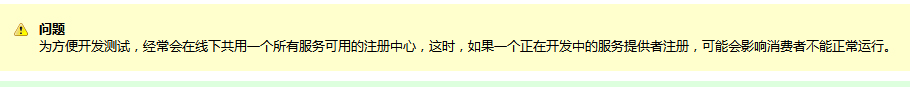
## 5.0 版本号处理服务升级



## 5.1 点对点直连(开发测试)

url="dubbo://www.gtp.test.com:20880"

## 5.2 只订阅(开发测试)

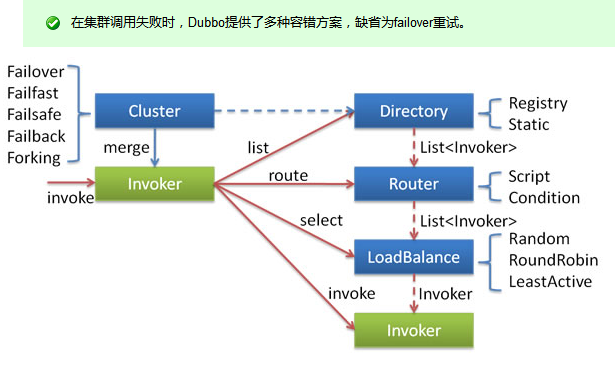




## 5.3 启动时服务检查

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，可以设置 <dubbo:reference interface="com.foo.BarService" check="false" />

## 5.4 集群容错



各节点关系：

* 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<Invoker>，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。

集群模式配置，

|  |
| --- |
| <dubbo:service cluster="failsafe" /> |

|  |
| --- |
| <dubbo:reference cluster="failsafe" /> |

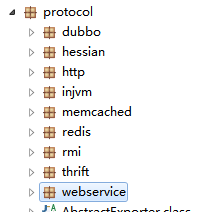
可配置的策略如下

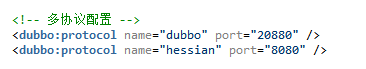


## 5.5 集群负载均衡



## 5.6 支持的协议

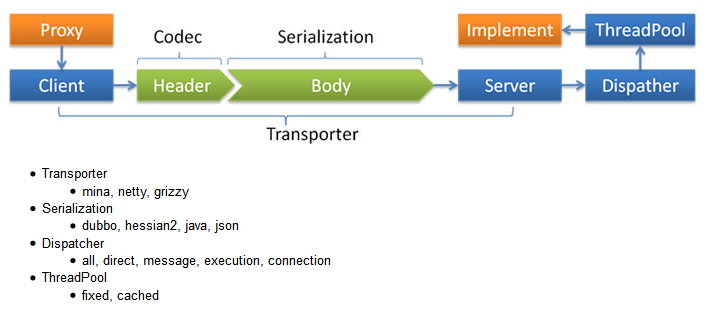




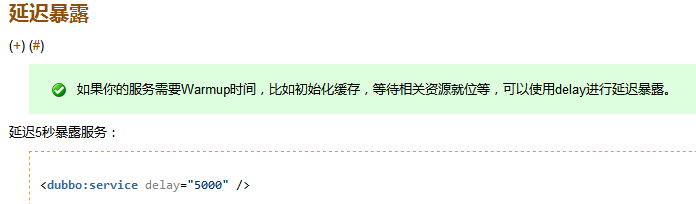
协议说明

Dubbo缺省协议采用单一长连接和NIO异步通讯，*适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况*

|  |  |
| --- | --- |
|  | Dubbo缺省协议不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低。 |



## 5.7 服务延迟暴露

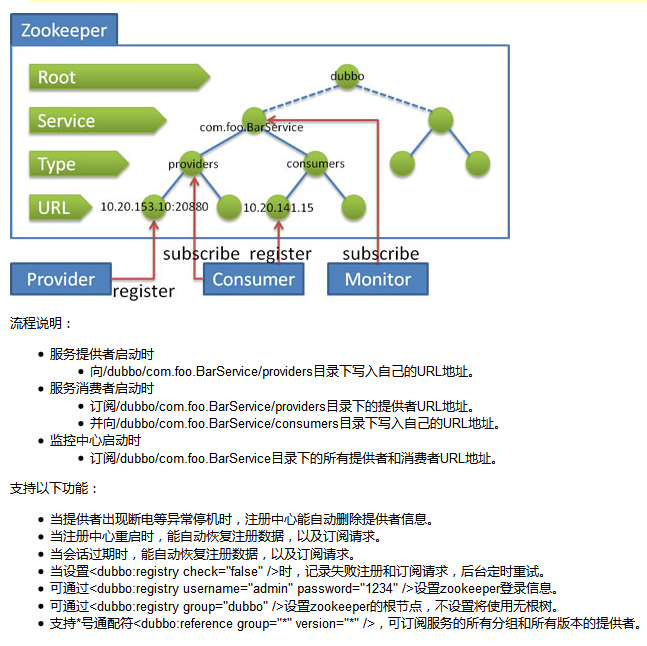


(**Spring2.x初始化死锁问题**)

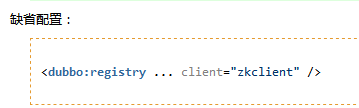
## 5.8 日志适配



## 5.9 zk注册中心



从2.2.0版本开始缺省为zkclient实现，以提升zookeeper客户端的健状性.



或

<dubbo:registry ... client="curator" />

<dependency>

    <groupId>com.github.sgroschupf</groupId>

    <artifactId>zkclient</artifactId>

    <version>0.1</version>

</dependency>

<dependency>

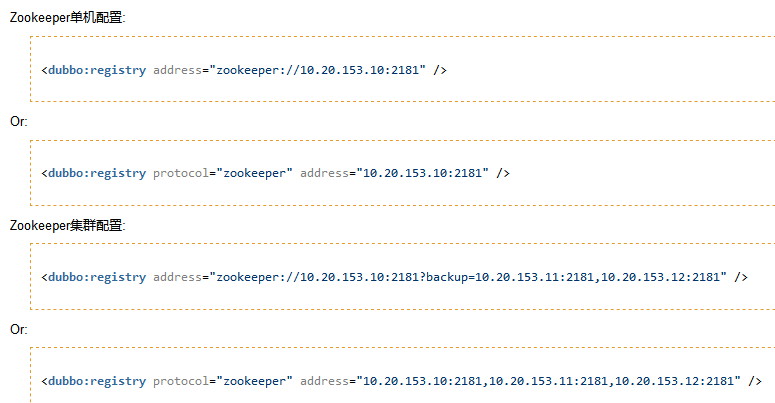
    <groupId>com.netflix.curator</groupId>

    <artifactId>curator-framework</artifactId>

    <version>1.1.10</version>

</dependency>

注意



# 6. 配置参考手册

## 6.1 dubbo.service







## 6.2 dubbo:reference





## 6.3 dubbo:protocol



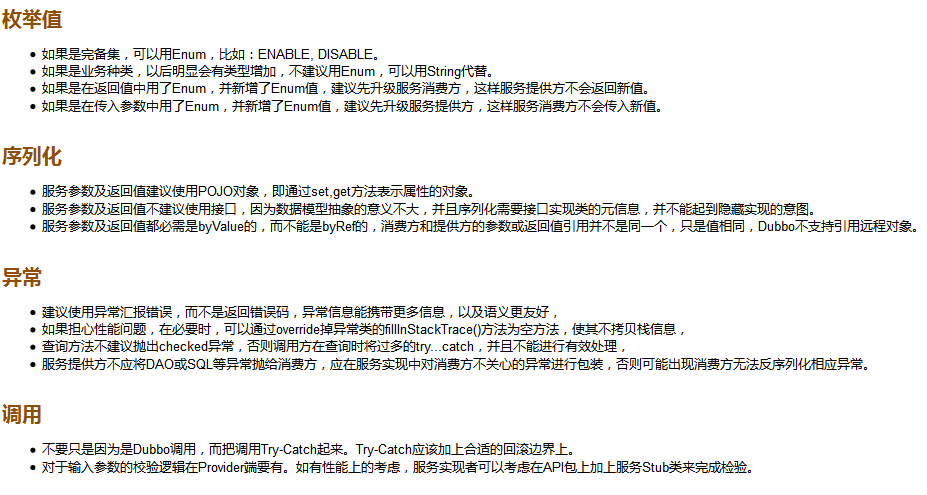


## 6.4 dubbo:registry



# 7. 服务化建议





# 8. 推荐用法

([+](http://dubbo.io/Recommended+Usage-zh.htm)) ([#](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%E6%8E%A8%E8%8D%90%E7%94%A8%E6%B3%95))

在Provider上尽量多配置Consumer端属性

原因如下：

作服务的提供者，比服务使用方更清楚服务性能参数，如调用的超时时间，合理的重试次数，等等

在Provider配置后，Consumer不配置则会使用Provider的配置值，即Provider配置可以作为Consumer的缺省值。  
否则，Consumer会使用Consumer端的全局设置，这对于Provider**不可控的**，并且往往是**不合理的**。  
PS: 配置的覆盖规则：1) 方法级配置别优于接口级别，即小Scope优先 2) Consumer端配置 优于 Provider配置 优于 全局配置，最后是Dubbo Hard Code的配置值（见配置文档）  
配置的覆盖规则详见： [Dubbo配置参考手册](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-ConfigurationReference)

Provider上尽量多配置Consumer端的属性，让Provider实现者一开始就思考Provider服务特点、服务质量的问题。

示例：

|  |
| --- |
| <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref="helloService"      timeout="300" retry="2" loadbalance="random" actives="0"  />    <dubbo:service interface="com.alibaba.hello.api.WorldService" version="1.0.0" ref="helloService"      timeout="300" retry="2" loadbalance="random" actives="0" >      <dubbo:method name="findAllPerson" timeout="10000" retries="9" loadbalance="leastactive" actives="5" />  <dubbo:service/> |

在Provider可以配置的Consumer端属性有：

**timeout**，方法调用超时

**retries**，失败重试次数，缺省是2（表示加上第一次调用，会调用3次）

**loadbalance**，负载均衡算法（有多个Provider时，如何挑选Provider调用），缺省是随机（random）。  
还可以有轮训(roundrobin)、最不活跃优先（leastactive，指从Consumer端并发调用最好的Provider，可以减少的反应慢的Provider的调用，因为反应更容易累积并发的调用）

**actives**，消费者端，最大并发调用限制，即当Consumer对一个服务的并发调用到上限后，新调用会Wait直到超时。  
在方法上配置（dubbo:method ）则并发限制针对方法，在接口上配置（dubbo:service），则并发限制针对服务。

详细配置说明参见：[Dubbo配置参考手册](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-ConfigurationReference)

Provider上配置合理的Provider端属性

|  |
| --- |
| <dubbo:protocol threads="200" />    <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref="helloService"      executes="200" >      <dubbo:method name="findAllPerson" executes="50" />  </dubbo:service> |

Provider上可以配置的Provider端属性有：

**threads**，服务线程池大小

**executes**，一个服务提供者并行执行请求上限，即当Provider对一个服务的并发调用到上限后，新调用会Wait（Consumer可能到超时）。在方法上配置（dubbo:method ）则并发限制针对方法，在接口上配置（dubbo:service），则并发限制针对服务。

配置上管理信息

目前有**负责人**信息和**组织**信息（用于区分站点）。

有问题时便于的找到服务的负责人，至少写**两个人**以便备份。

负责人和组织的信息可以在注册中心的上看到。

示例：

**应用配置负责人、组织**

|  |
| --- |
| <dubbo:application owner=”ding.lid,william.liangf” organization=”intl” /> |

**service配置负责人**

|  |
| --- |
| <dubbo:service owner=”ding.lid,william.liangf” /> |

**reference配置负责人**

|  |
| --- |
| <dubbo:reference owner=”ding.lid,william.liangf” /> |

dubbo:service、dubbo:reference没有配置负责人，则使用dubbo:application设置的负责人。

配置上Dubbo缓存文件

配置方法如下：

**提供者列表缓存文件**

|  |
| --- |
| <dubbo:registry file=”${user.home}/output/dubbo.cache” /> |

注意：

文件的路径，应用可以根据需要调整，保证这个文件不会在发布过程中被清除。

如果有多个应用进程注意不要使用同一个文件，避免内容被覆盖。

这个文件会缓存：

注册中心的列表

服务提供者列表

有了这项配置后，当应用重启过程中，Dubbo注册中心不可用时则应用会从这个缓存文件读取服务提供者列表的信息，进一步保证应用可靠性。

监控配置

1. 使用固定端口暴露服务，而不要使用随机端口

这样在注册中心推送有延迟的情况下，消费者通过缓存列表也能调用到原地址，保证调用成功。

3. 使用Dragoon的http监控项监控注册中心上服务提供方

Dragoon监控服务在注册中心上的状态：[http://dubbo-reg1.hst.xyi.cn.alidc.net:8080/status/com.alibaba.morgan.member.MemberService:1.0.5](javascript:if(confirm(%27http://dubbo-reg1.hst.xyi.cn.alidc.net:8080/status/com.alibaba.morgan.member.MemberService:1.0.5%20%20/n/nThis%20file%20was%20not%20retrieved%20by%20Teleport%20Ultra,%20because%20it%20is%20addressed%20on%20a%20domain%20or%20path%20outside%20the%20boundaries%20set%20for%20its%20Starting%20Address.%20%20/n/nDo%20you%20want%20to%20open%20it%20from%20the%20server?%27))window.location=%27http://dubbo-reg1.hst.xyi.cn.alidc.net:8080/status/com.alibaba.morgan.member.MemberService:1.0.5%27)，确保注册中心上有该服务的存在。

4. 服务提供方，使用Dragoon的telnet或shell监控项

监控服务提供者端口状态：echo status | nc --i 1 20880 | grep OK | wc --l，其中的20880为服务端口

5. 服务消费方，通过将服务强制转型为EchoService，并调用$echo()测试该服务的提供者是可用

如 assertEqauls(“OK”, ((EchoService)memberService).$echo(“OK”));

不要使用dubbo.properties文件配置，推荐使用对应XML配置

Dubbo2中所有的配置项都可以Spring配置中，并且可以针对单个服务配置。

# 如完全不配置使用Dubbo缺省值，参见[Dubbo配置参考手册](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-ConfigurationReference)中的说明。

在Dubbo1中需要在dubbo.properties文件中的配置项，Dubbo2中配置示例如下：

1. 应用名

|  |
| --- |
| <dubbo:application name="myalibaba" > |

对应dubbo.properties中的Key名dubbo.application.name

2. 注册中心地址

|  |
| --- |
| <dubbo:registry address="11.22.33.44:9090" > |

对应dubbo.properties中的Key名dubbo.registry.address

3. 调用超时

可以在多个配置项设置超时，由上至下**覆盖**（即上面的优先），示例如下：

# 其它的参数（retries、loadbalance、actives等）的覆盖策略也一样。

提供者端特定方法的配置

|  |
| --- |
| <dubbo:service interface="com.alibaba.xxx.XxxService" >      <dubbo:method name="findPerson" timeout="1000" />  </dubbo:service> |

提供者端特定接口的配置

|  |
| --- |
| <dubbo:service interface="com.alibaba.xxx.XxxService" timeout="200" /> |

# timeout可以在多处设置，配置项及覆盖规则详见： [Dubbo配置参考手册](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-ConfigurationReference)

全局配置项值，对应dubbo.properties中的Key名dubbo.service.invoke.timeout

5. 服务提供者协议、服务的监听端口

|  |
| --- |
| <dubbo:protocol name="dubbo" port="20880" /> |

对应dubbo.properties中的Key名dubbo.service.protocol、dubbo.service.server.port

4. 服务线程池大小

|  |
| --- |
| <dubbo:protocol threads="100" /> |

对应dubbo.properties中的Key名dubbo.service.max.thread.threads.size

6. 消费者启动时，没有提供者是否抛异常Fast-Fail

|  |
| --- |
| <dubbo:reference interface="com.alibaba.xxx.XxxService" check="false" /> |

对应dubbo.properties中的Key名alibaba.intl.commons.dubbo.service.allow.no.provider