一、启动时check

dubbo提供了在服务启动时的一些检查机制，这个机制包括consumer端对服务提供者的检查 、 dubbo对注册中心的检查。

可以配置<dubbo:reference check = "false">

同时可以配置 全局的consumer配置 dubbo:consumer check = "false" 这样在consumer端注册的服务引用都不会去在启动时check是否有对应的服务注册。
这里注意文档中说的：

`dubbo.reference.check=false`，强制改变所有 reference 的 check 值，就算配置中有声明，也会被覆盖。

`dubbo.consumer.check=false`，是设置 check 的缺省值，如果配置中有显式的声明，如：`<dubbo:reference check="true"/>`，不会受影响。

二、超时配置

超时之后，会报一个wait return response这个错误
可以从官方的配置中看到，默认reference标签的超时是继承自dubbo consumer标签的timeout属性 是1000

超时时间有个覆盖策略：
（1）更精确优先（优先方法就）
（2）消费方优先

三、重试次数

注意重试次数 不包括第一次调用，比如retries写的是3，那么会最多调用4次。
并且重试也会load balance：如果有多个服务提供者，重试会尝试调用下一个服务提供者。这里有它自己的负载均衡

注意重试要做在幂等操作上。

这里可以启动多个provider-main方法，来暴露不同的端口服务，来模拟对应的多个provider

四、多版本功能

同一个服务提供者可以存在多个版本，比如1.0.0 2.0.0 这里可以使得消费者指定dubbo接口的版本，来实现平滑的版本升级。

dubbo:service标签可以定义version属性

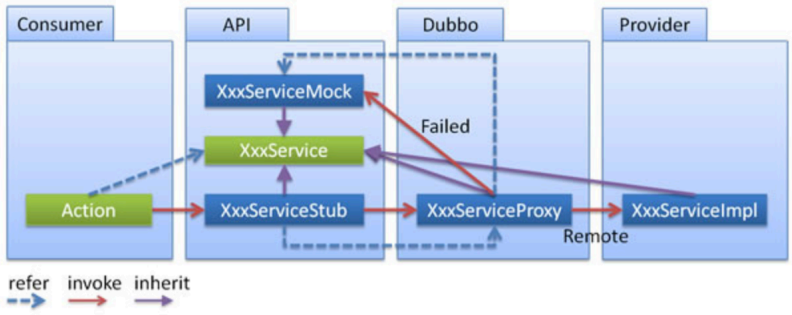
dubbo:reference标签可以指定引用的版本，当version="*"时，则说明随机引用dubbo接口对应的版本。

五、本地存根

本地存根可以在调用真正的远程服务之前，做一个本地存根的验证。
实现本店存根功能的类必须有一个有参数构造器去将代理对象传入到真正的执行方法中，这里就可以做一些本地存根中要做的逻辑。

本地存根

远程服务后，客户端通常只剩下接口，而实现全在服务器端，但提供方有些时候想在客户端也执行部分逻辑，比如：做 ThreadLocal 缓存，提前验证参数，调用失败后伪造容错数据等等，此时就需要在 API 中带上 Stub，客户端生成 Proxy 实例，会把 Proxy 通过构造函数传给 Stub [1]，然后把 Stub 暴露给用户，Stub 可以决定要不要去调 Proxy。



本地存根的stub类必须定义在接口的包里，才能实现stub功能