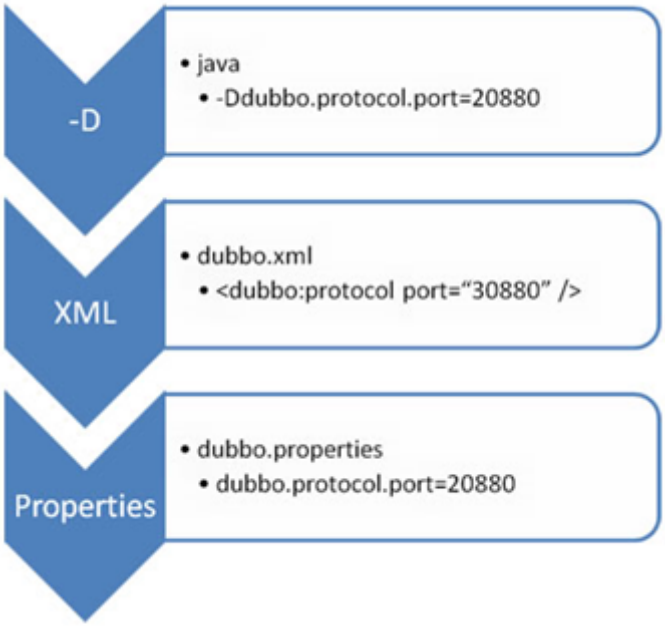
dubbo笔记（2）dubbo配置

一、配置原则

配置顺序



- JVM 启动 -D 参数优先，这样可以使用户在部署和启动时进行参数重写，比如在启动时需改变协议的端口。
- XML 次之，如果在 XML 中有配置，则 dubbo.properties 中的相应配置项无效。
- Properties 最后，相当于缺省值，只有 XML 没有配置时，dubbo.properties 的相应配置项才会生效，通常用于共享公共配置，比如应用名。

二、启动时检查

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，阻止 Spring 初始化完成，以便上线时，能及早发现问题，默认 `check="true"`。

可以通过 `check="false"` 关闭检查，比如，测试时，有些服务不关心，或者出现了循环依赖，必须有一方先启动。

另外，如果你的 Spring 容器是懒加载的，或者通过 API 编程延迟引用服务，请关闭 check，否则服务临时不可用时，会抛出异常，拿到 null 引用，如果 `check="false"`，总是会返回引用，当服务恢复时，能自动连上。

通过 spring 配置文件

关闭某个服务的启动时检查 (没有提供者时报错)：

```
<dubbo:reference interface="com.foo.BarService" check="false" />
```

关闭所有服务的启动时检查 (没有提供者时报错)：

```
<dubbo:consumer check="false" />
```

关闭注册中心启动时检查 (注册订阅失败时报错)：

```
<dubbo:registry check="false" />
```

配置的含义

`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。

`dubbo.registry.check=false`，前面两个都是指订阅成功，但提供者列表是否为空是否报错，如果注册订阅失败时，也允许启动，需使用此选项，将在后台定时重试。

三、重试次数

失败自动切换，当出现失败，重试其它服务器，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)。

重试次数配置如下：

```
<dubbo:service retries="2" />
```

或

```
<dubbo:reference retries="2" />
```

或

```
<dubbo:reference>
    <dubbo:method name="findFoo" retries="2" />
</dubbo:reference>
```

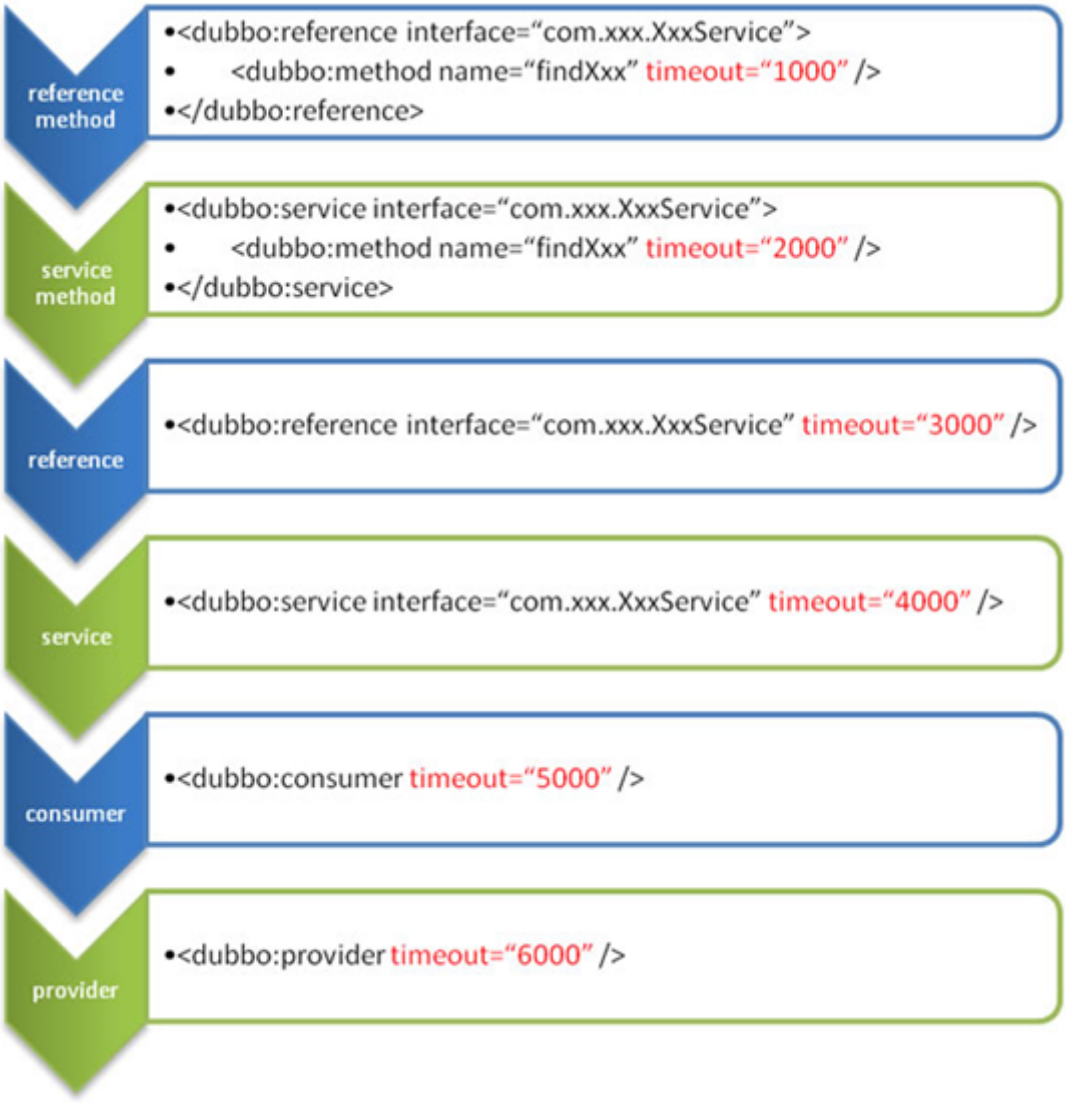
四、超时时间

dubbo推荐在Provider上尽量多配置Consumer端属性：

> 1、作为服务的提供者，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
>
> 2、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

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) Scope相同时，Consumer端配置 优于 Provider配置



五、多版本

当一个接口实现，出现不兼容升级时，可以用版本号过渡，版本号不同的服务相互间不引用。

可以按照以下的步骤进行版本迁移：

1. 在低压力时间段，先升级一半提供者为新版本
2. 再将所有消费者升级为新版本
3. 然后将剩下的一半提供者升级为新版本

老版本服务提供者配置：

```
<dubbo:service interface="com.foo.BarService" version="1.0.0" />
```

新版本服务提供者配置：

```
<dubbo:service interface="com.foo.BarService" version="2.0.0" />
```

老版本服务消费者配置：

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="1.0.0"
/>
```

新版本服务消费者配置：

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="2.0.0"
/>
```

如果不需要区分版本，可以按照以下的方式配置：

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="*" />
```

六、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，然后把 Stub 暴露给用户，Stub 可以决定要不要去调 Proxy。

在消费者 spring 配置文件中按以下方式配置：

```
<dubbo:reference interface="com.foo.BarService" stub="com.foo.BarServiceStub" />
```

提供 Stub 的实现：

```
package com.foo;
public class BarServiceStub implements BarService {
    private final BarService barService;

    // 构造函数传入真正的远程代理对象
    public BarServiceStub(BarService barService){
        this.barService = barService;
    }

    public String sayHello(String name) {
        // 此代码在客户端执行，你可以在客户端做ThreadLocal本地缓存，或预先验证参数是否合法，
等等
        try {
            return barService.sayHello(name);
        } catch (Exception e) {
            // 你可以容错，可以做任何AOP拦截事项
            return "容错数据";
        }
    }
}
```