分布式服务治理Dubbo的常用配置及源码分析

- 1、多版本支持
  - 设置不同版本的目的，就是要考虑到接口升级以后带来的兼 容问题。在Dubbo中配置不同版本的接口，会在Zookeeper 地址中有多个协议url的体现，具体内容如下
    - dubbo://192.168.11.1:20880%2Fcom.gupaoedu.dubbo.IGpHello%3Fanyhost%3Dtrue%26application%3Dhelloworldapp%26dubbo%3D2.5.6%26generic%3Dfalse%26interface%3Dcom.gupaoedu.dubbo.IGpHello%26methods%3DsayHello%26pid%3D60700%26revision%3D1.0.0%26side%3Dprovider%26timestamp%3D1529498478644%26version%3D1.0.0
    - dubbo://192.168.11.1%3A20880%2Fcom.gupaoedu.dubbo.IGpHello2%3Fanyhost%3Dtrue%26application%3Dhelloworldapp%26dubbo%3D2.5.6%26generic%3Dfalse%26interface%3Dcom.gupaoedu.dubbo.IGpHello%26methods%3DsayHello%26pid%3D60700%26revision%3D1.0.1%26side%3Dprovider%26timestamp%3D1529498488747%26version%3D1.0.1
- 2、主机绑定
  - 在发布一个Dubbo服务的时候，会生成一个dubbo://ip:port 的协议地址，那么这个IP 是根据什么生成的呢？大家可以在ServiceConfig.java代码中找到如下代码;可以发现，在生成绑定的主机的时候，会通过一层一层的判断，直到获取到合法的 ip 地址。

```
1. NetUtils.isInvalidLocalHost(host),   从配置文件中获取host
2. host = InetAddress.getLocalHost().getHostAddress();
3. Socket socket = new Socket();
    try {
        SocketAddress addr = new
        InetSocketAddress(registryURL.getHost(),
        registryURL.getPort());
        socket.connect(addr, 1000);
        host = socket.getLocalAddress().getHostAddress();
        break;
    } finally {
    try {
        socket.close();
    } catch (Throwable e) {}
}
```

```
public static String getLocalHost(){
    InetAddress address = getLocalAddress();
    return address == null ? LOCALHOST : address.getHostAddress();
}
```

- 3、集群容错
  - 什么是容错机制？容错机制指的是某种系统控制在一定范围内的一种允许或包容犯错情况的发生，举个简单例 子，我们在电脑上运行一个程序，有时候会出现无响应的情况，然后系统会弹出一个提示框让我们选择，是立即结 束还是继续等待，然后根据我们的选择执行对应的操作， 这就是“容错”。
  - 在分布式架构下，网络、硬件、应用都可能发生故障，由 于各个服务之间可能存在依赖关系，如果一条链路中的其中一个节点出现故障，将会导致雪崩效应。为了减少某一 个节点故障的影响范围，所以我们才需要去构建容错服务， 来优雅的处理这种中断的响应结果
  - 6种容错机制

- 1）failsafe失败安全
- 2）failover（默认）失败重试其他服务器；retries（2）
- 3）failfast快速失败，失败后立马报错
- 4）failback 失败后自动恢复
- 5）forking forks 设置并行数
- 6）broadcast 广播，任意一台报错，则执行的方法报错
- 配置地方

```
<!-- 声明需要暴露的服务接口 -->
<dubbo:reference id="demoService" interface="com.gupaoedu.dubbo.IGpHello"
                 registry="zookeeper" version="1.0.0"
cluster="failsafe"/>
```

- 4、服务降级
  - 降级的目的是为了保证核心服务可用。
  - 降级可以有几个层面的分类： 自动降级和人工降级； 按 照功能可以分为：读服务降级和写服务降级；
    - 1. 对一些非核心服务进行人工降级，在大促之前通过降级 开关关闭哪些推荐内容、评价等对主流程没有影响的功 能
    - 2. 故障降级，比如调用的远程服务挂了，网络故障、或者 RPC服务返回异常。 那么可以直接降级，降级的方案比 如设置默认值、采用兜底数据（系统推荐的行为广告挂 了，可以提前准备静态页面做返回）等等
    - 3. 限流降级，在秒杀这种流量比较集中并且流量特别大的 情况下，因为突发访问量特别大可能会导致系统支撑不 了。这个时候可以采用限流来限制访问量。当达到阀值 时，后续的请求被降级，比如进入排队页面，比如跳转 到错误页（活动太火爆，稍后重试等）
  - 降级方式：Mock（客户端的一个策略）
    - dubbo的降级方式： Mock 实现步骤
      - 1. 在client端创建一个TestMock类，实现对应IGpHello 的接口（需要对哪个接口进行mock，就实现哪个）， 名称必须以Mock结尾
      - 2. 在client端的xml配置文件中，添加如下配置，增加一 个mock属性指向创建的TestMock
      - 3. 模拟错误（设置 timeout），模拟超时异常，运行测试代码即可访问到TestMock这个类。当服务端故障解除 以后，调用过程将恢复正常

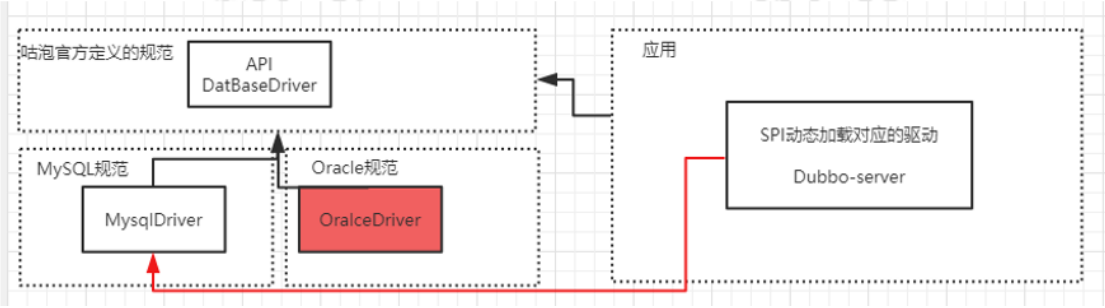
```
<!-- 声明需要暴露的服务接口 -->
<dubbo:reference id="demoService"
                 interface="com.gupaoedu.dubbo.IGpHello"
                 registry="zookeeper"

                 mock="com.gupaoedu.dubbo.TestMock" timeout="50"/>
```

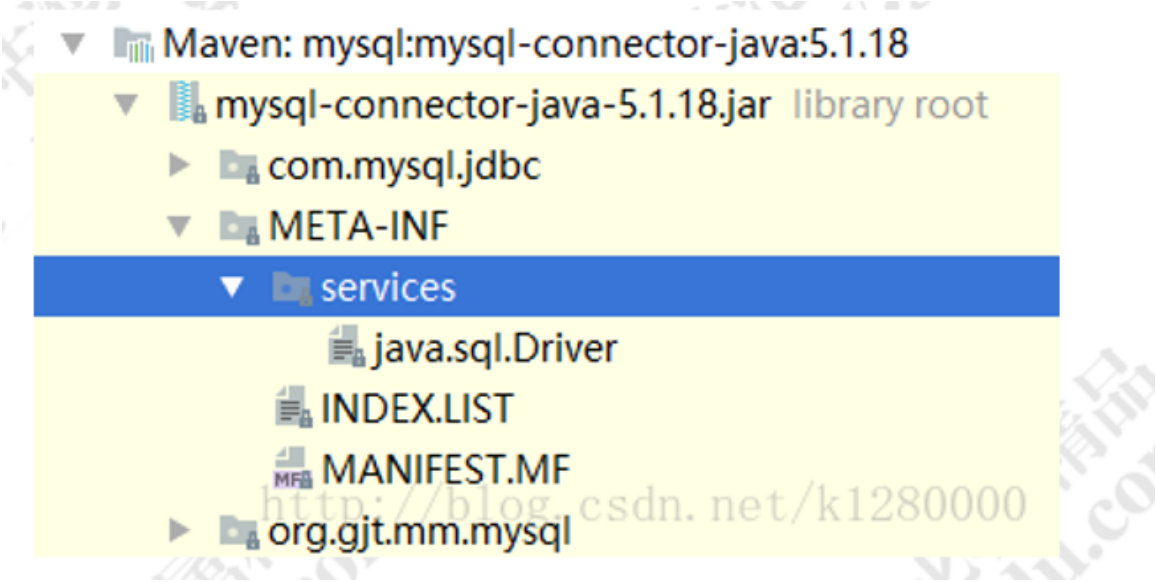
- 5、配置优先级别
  - 客户端会优于服务端，
  - 1、可以细粒度到方法级别、然后是接口，最后是全局配置
  - 2、如果级别是一样的，客户端优先
  - retires、LoadBanlance、cluster（客户端）、timeout（服务端）
  - 以 timeout 为例，显示了配置的查找顺序，其它 retries, loadbalance等类似。 1. 方法级优先，接口级次之，全局配置再次之。 2. 如果级别一样，则消费方优先，提供方次之。 其中，服务提供方配置，通过URL经由注册中心传递给消 费方。
  - 建议由服务提供方设置超时，因为一个方法需要执行多长 时间，服务提供方更清楚，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。

- 6、什么是SPI
  - 在Dubbo中，SPI是一个非常核心的机制，贯穿在几乎所 有的流程中。搞懂这块内容，是接下来了解Dubbo更多源 码的关键因素，Dubbo是基于Java原生SPI机制思想的一个改进。所以， 先从 JAVA SPI机制开始了解什么是 SPI 以后再去学习 Dubbo的SPI，就比较容易了
  - JAVA 的SPI
    - SPI全称（ service provider interface ） ，是JDK内置的一种 服务提供发现机制，目前市面上有很多框架都是用它来做服 务的扩展发现，大家耳熟能详的如 JDBC、日志框架都有用 到；
    - 简单来说，它是一种动态替换发现的机制。举个简单的例子， 如果我们定义了一个规范，需要第三方厂商去实现，那么对 于我们应用方来说，只需要集成对应厂商的插件，既可以完 成对应规范的实现机制。 形成一种插拔式的扩展手段。
    - 实现一个SPI机制，实现的代码的流程图如下



    - 代码演示
    - SPI规范总结
      - 实现SPI，就需要按照SPI本身定义的规范来进行配置，SPI 规范如下
      - 1. 需要在classpath下创建一个目录，该目录命名必须是： META-INF/services
      - 2. 在该目录下创建一个properties文件，该文件需要满足 以下几个条件 a) 文件名必须是扩展的接口的全路径名称 b) 文件内部描述的是该扩展接口的所有实现类 c) 文件的编码格式是 UTF-8
      - 3. 通过java.util.ServiceLoader的加载机制来发现
    - SPI的实际应用
      - SPI 在很多地方有应用，大家可以看看最常用的 java.sql.Driver 驱动。JDK 官方提供了 java.sql.Driver 这个 驱动扩展点，但是你们并没有看到 JDK 中有对应的 Driver 实现。 那在哪里实现呢？ 以连接Mysql为例，我们需要添加mysql-connector-java 依赖。然后，你们可以在这个jar包中找到SPI的配置信息。
      - 如下图，所以java.sql.Driver由各个数据库厂商自行实现。 这就是SPI的实际应用。当然除了这个意外，大家在spring 的包中也可以看到相应的痕迹



- SPI的缺点
  - 1. JDK标准的SPI会一次性加载实例化扩展点的所有实现， 什么意思呢？就是如果你在 META-INF/service 下的文件 里面加了N个实现类，那么JDK启动的时候都会一次性全 部加载 。那么如果有的扩展点实现初始化很耗时或者如果 有些实现类并没有用到，那么会很浪费资源
  - 2. 如果扩展点加载失败，会导致调用方报错，而且这个错 误很难定位到是这个原因
- Dubbo优化后的SPI实现
  - 基于Dubbo提供的SPI规范实现自己的扩展
    - 在了解Dubbo的SPI机制之前，先通过一段代码初步了解Dubbo的实现方式，这样，我们就能够形成一个对比，得 到这两种实现方式的差异 《代码实现，参考视频直播+课后的视频，这里偷懒不写了》
  - Dubbo的SPI机制规范
    - 大部分的思想都是和 SPI 是一样，只是下面两个地方有差 异。 1. 需要在 resource 目录下配置 META-INF/dubbo 或者 META-INF/dubbo/internal或者META-INF/services，并基 于SPI接口去创建一个文件 2. 文件名称和接口名称保持一致，文件内容和SPI有差异， 内容是KEY对应Value

- 7、源码阅读之Dubbo中的SPI机制分析