crazyhzm 265

Dubbo源码解析（四十三）2.7新特性

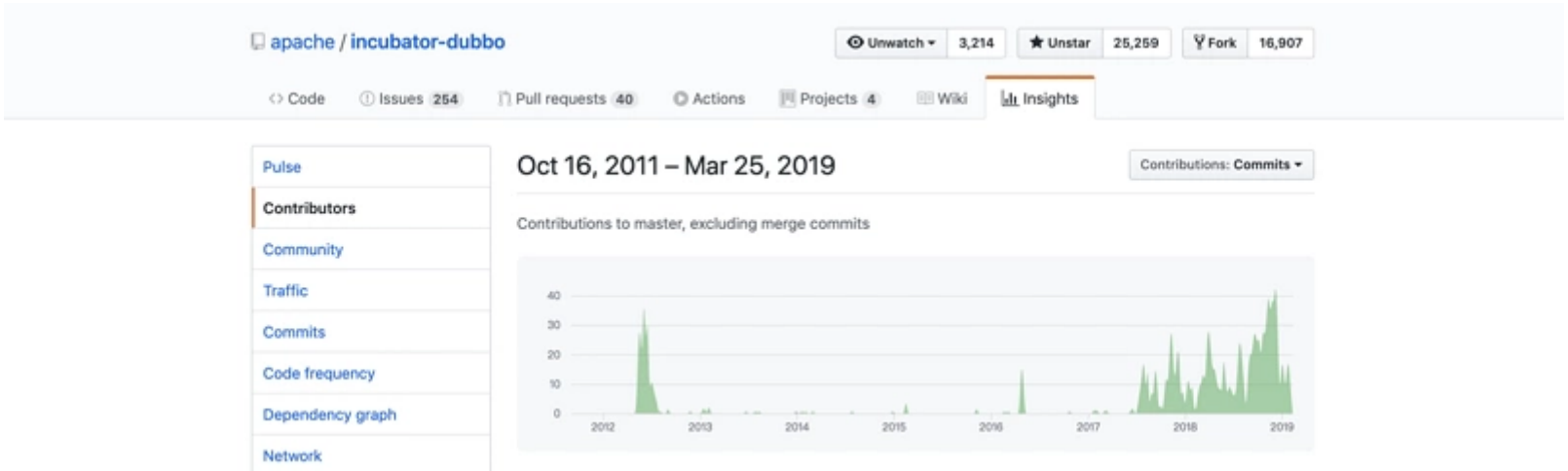
dubbo java 阅读约 11 分钟

DUBBO——2.7大揭秘

> 目标：了解2.7的新特性，以及版本升级的引导。

前言

我们知道Dubbo在2011年开源，停止更新了一段时间。在2017 年 9 月 7 日，Dubbo 悄悄的在 GitHub 发布了 2.5.4 版本。随后，版本发布的非常迅速，Dubbo项目被重启了，经过大半年的更新，在2018年2月15日，Dubbo 获得了 14 张赞成票，在无弃权和反对票的情况下，正式通过投票，顺利成为 Apache 基金会孵化项目。现在的Dubbo社区非常活跃，版本进度也非常的快。



从上图就可以看出dubbo现在的活跃度。

现在dubbo项目有以下几个分支：

1. 2.5.x：该分支在近期投票决定不再维护。
2. 2.6.x：该分支现在还在维护中，包名前缀是com.alibaba，也是贡献给 Apache 之前的版本。
3. 2.7.1-release：一个临时分支。
4. 3.x-dev：将以 Streaming 为内核，重点的改变在服务治理和编程模型上。具体我也还没有深入研究，我也会跟踪该分支的变动，敬请期待吧。
5. master：目前版本是2.7.x，包名前缀是：org.apache，也是 Dubbo 贡献给 Apache 的开发版本，接下来的分析也会在2.7.1上进行分析。

关注dubbo社区的朋友应该也知道在2019.3.23在南京举办了Meetup，其中有一个专题就是讲2.7新特性介绍。我就在分享者的基础上讲解一下自己的理解。

正文

(一)JDK版本

在所需的最小JDK版本从以前的1.6变成了1.8。

(二)包重命名

com.alibaba.dubbo - > org.apache.dubbo

(三)异步支持优化

我们知道dubbo协议本身支持三种发送请求方式：

1. 单向发送：执行方法不需要返回结果
2. 同步发送：执行方法后，等待结果返回，否则一直阻塞.
3. 异步发送：也就是当我发送调用后，我不阻塞等待结果，直接返回，将返回的future保存到上下文，方便后期使用。在异步发送中有两种方式分别是
    1. future：当请求有响应后，通过future.get()来获得响应结果，但是future.get()会导致线程阻塞，future从RpcContext获取。
    2. callback：设置一个回调线程，当接收到响应时，自动执行，不会对当前线程造成阻塞，自定义ResponseFuture支持callback。

2.6.x版本的异步方式提供了一些异步能力，包括Consumer端异步调用、参数回调、事件通知等。但当前的异步方式存在以下问题：

- Future获取方式不够直接，只能在RpcContext中进行获取；
- Future只支持阻塞式的get()接口获取结果。
- Future接口无法实现自动回调，而自定义ResponseFuture虽支持callback回调但支持的异步场景有限，如不支持Future间的相互协调或组合等；
- 不支持Provider端异步

具体的可以参考该文章dubbo源码解析（二十四）远程调用——dubbo协议中的源码分析来理解其中存在的问题。

那么在2.7.x版本，由于JDK版本升级到了1.8，引入了JDK1.8 中的CompletableFuture接口，CompletableFuture支持 future 和 callback 两种调用方式。关于CompletableFuture怎么被运用到dubbo中我会在后续的文章介绍。引入该接口后，做了以下优化：

- 支持Provider端异步
- 支持直接定义返回CompletableFuture的服务接口。通过这种类型的接口，我们可以更自然的实现Consumer、Provider端的异步编程。

```
public interface AsyncService {
    CompletableFuture<String> sayHello(String name);
}
```

- 如果你不想将接口的返回值定义为Future类型，或者存在定义好的同步类型接口，则可以额外定义一个异步接口并提供Future类型的方法。

```
public interface GreetingsService {
    String sayHi(String name);
}
```

```
@AsyncFor(GreetingsService.class)
public interface GrettingServiceAsync extends GreetingsService {
    CompletableFuture<String> sayHiAsync(String name);
}
```

- 如果你的原始接口定义不是Future类型的返回值，Provider端异步也提供了类似Servlet3.0里的Async Servlet的编程接口:

  `RpcContext.startAsync()`

```
public interface AsyncService {
    String sayHello(String name);
}
```

```
public class AsyncServiceImpl implements AsyncService {
    public String sayHello(String name) {
        final AsyncContext asyncContext = RpcContext.startAsync();
        new Thread(() -> {
            asyncContext.write("Hello " + name + ", response from provider.");
        }).start();
        return null;
    }
}
```

- 异步过滤器链回调。

具体的实现原理我在后续文章中结合源码来讲解，**注意**：这些改动都仅仅支持dubbo协议。

(四)元数据改造

我们知道2.7以前的版本只有注册中心，注册中心的URL有数十个key/value的键值对，包含了一个服务所有的元数据。在越来越多的功能被增加，元数据也变得异常庞大，就出现了下面的问题：

- 注册中心存储的URL过长：这会导致存储的压力骤增，数据庞大导致在修改元数据后的通知效率也下降，并且增加了消费者对于元数据解析的压力，尤其是在大规模场景下的内存增长显著
- 注册中心承担了过多的服务治理配置的功能：初始配置的同步、存储各种运行期配置规则加剧了注册中心的压力，配置规则的灵活性也有所限制，阻碍了市场上的一些优秀微服务配置中心的集成和扩展。
- 属性的功能定位不清晰：methods，pid，owner虽然是为了查询服务而注册的属性，但是这些简陋的信息很难满足查询服务治理需求，所以需要更加丰富的注册数据。例如methods，虽然方法列表的内容已经很长，但是在ops开发服务测试/mock功能时，发现需要的方法签名等数据还是无法获取。

针对以上问题，在2.7中，将URL中的元数据划分了三个部分：

- 元数据信息：接口的完整定义，包含接口名，接口所含的方法，以及方法所含的出入参信息。对于服务测试和服务mock有很重要作用。
- 执行链路上数据：需要将参数从provider端传递给consume端，让consume端感知的到，比如token、timeout等
- 服务自己持有的配置&Ops需求：只有provider端自己需要或者consume端自己需要的数据，比如executes、document等

改造后，分别形成三大中心：

- 注册中心：理想情况下，注册中心将只用于关键服务信息（核心链路）的同步，进一步减轻注册中心的存储压力，提高地址同步效率，同时缓解当前由于URL冗余在大规模推送时造成的Consumer端内存计算压力。
- 配置中心：解决当前配置和地址信息耦合的问题，通过抽象动态配置层，让开发者可以对接微服务场景下更常用的、更专业的配置中心，如Nacos, Apollo, Consul, Etcd等；提供更灵活的、更丰富的配置规则，包括服务、应用不同粒度的配置，更丰富的路由规则，集中式管理的动态参数规则等
- 服务查询治理中心：对于纯粹的服务查询相关的数据，包括Consumer的服务订阅数据，往往都是注册后不可变的并且不需要节点间的同步，如当前URL可以看到的methods、owner等key以及所有的Consumer端URL，目前支持 redis（推荐），zookeeper, 将作为Dubbo-Admin支持的服务测试，模拟和其他服务治理功能的基础。

(五)服务治理规则增强

路由规则的增强

Dubbo 提供了具有一定扩展性的路由规则，其中具有代表性的是条件路由和脚本路由。2.6.x及以下版本存在的问题：

- 路由规则存储在注册中心
- 只支持服务粒度的路由，应用级别无法定义路由规则
- 支持路由缓存，但基本不具有扩展性
- 一个服务或应用允许定义多条路由规则，服务治理无法管控
- 实现上，每条规则生成一个Router实例并动态加载

在2.7.x版本中，对路由规则做了增强：

- 丰富的路由规则。
  - 条件路由：支持应用程序级别和服务级别条件。
  - 标记路由：新引入以更好地支持流量隔离，例如灰色部署

配置中心对服务治理的加成

- 将治理规则与注册表分离，也就是出现了配置中心，使配置中心更容易扩展。有Apollo和Zookeeper，2.7.1还支持了consul和etcd。
- 应用程序级动态配置支持。
- 使用YAML作为配置语言，更易于阅读和使用

(六)新增配置中心

配置中心（v2.7.0）在Dubbo中承担两个职责：

- 外部化配置：启动配置的集中式存储 （简单理解为dubbo.properties的外部化存储）外部化配置目的之一是实现配置的集中式管理，这部分业界已经有很多成熟的专业配置系统如Apollo, Nacos等，Dubbo所做的主要是保证能配合这些系统正常工作。外部化配置和其他本地配置在内容和格式上并无区别，可以简单理解为`dubbo.properties`的外部化存储，配置中心更适合将一些公共配置如注册中心、元数据中心配置等抽取以便做集中管理
- 服务治理：服务治理规则的存储与通知。

配置的操作可以查看官方文档，由于现在dubbo支持多种配置方式，所以这里需要强调的是配置覆盖的优先级，从上至下优先级依此降低：

(七)序列化扩展

新增了Protobuf序列化支持。

(八)其他

其他的bug修复以及一些小细节优化请查看github上的Issues或者PR。

后记

> 升级2.7.0的引导请查看以下链接:http://dubbo.apache.org/zh-cn...

该文章讲解了dubbo2.7的新特性，现在2.7.1已经发布，有兴趣的可以去看看2.7.1新增了什么。下一篇我就先从源码的角度来讲讲这个异步化的改造。

阅读 1.1k • 更新于 11月8日

crazyhzm

265

关注作者

0 条评论

得票 • 时间

撰写评论 ...

提交评论

推荐阅读

dubbo源码解析（一）Hello,Dubbo

你好，dubbo，初次见面，我想和你交个朋友。先给出一套官方的说法：ApacheDubbo是一款高性能、轻量级基于Java的RPC开源...

CrazyHzm • 阅读 633

SDWebImage源码解析(四)

这篇博文将分析SDWebImageDownloader和SDWebImageDownloaderOperation。SDWebImage通过这两个类处理图片的网络加载...

huang303513 • 阅读 13

Dubbo 源码分析 - SPI 机制

SPI全称为ServiceProviderInterface，是Java提供的一种服务发现机制。SPI的本质是将接口实现类的全限定名配置在文件中，并由服...

coolblog • 阅读 250

dubbo源码解析（十五）远程通信——Mina

ApacheMINA是一个网络应用程序框架，可帮助用户轻松开发高性能和高可扩展性的网络应用程序。它通过JavaNIO在各种传输（...

• 阅读 721

dubbo源码解析（二）Dubbo扩展机制SPI

前一篇文章《dubbo源码解析（一）Hello,Dubbo》是对dubbo整个项目大体的介绍，而从这篇文章开始，我将会从源码来解读dub...

CrazyHzm • 阅读 355

Spring-boot+Dubbo应用启停源码分析

背景介绍DubboSpringBoot工程致力于简化DubboRPC框架在SpringBoot应用场景的开发。同时也整合了SpringBoot特性：你有没...

阿里云云栖社区 • 阅读 13

dubbo源码解析——概要篇

这次源码解析借鉴《肥朝》前辈的dubbo源码解析，进行源码学习。总结起来就是先总体,后局部.也就是先把需要注意的概念先抛...

• 阅读 631