



# 【Dubbo3.0新特性】集成RSocket,新增响应式支持

© 2019 徐靖峰

Powered by [Hexo](#). Theme by [PPOffice](#)

原创



2019-04-11



RPC

## 响应式编程

响应式编程现在是现在一个很热的话题。响应式编程让开发者更方便地编写高性能的异步代码，关于响应式编程更详细的信息可以参考 <http://reactivex.io/>。很可惜，在之前很长一段时间里，Dubbo 并不支持响应式编程，简单来说，Dubbo 不支持在 rpc 调用时，使用 Mono/Flux 这种流对象（reactive-stream 中流的概念），给用户使用带来了不便。

RSocket 是一个支持 reactive-stream 语义的开源网络通信协议，它将 reactive 语义的复杂逻辑封装了起来，使得上层可以方便实现网络程序。RSocket 详细资料：<http://rssocket.io/>。

Dubbo 在 3.0.0-SNAPSHOT 版本里基于 RSocket 对响应式编程提供了支持，用户可以在请求参数和返回值里使用 Mono 和 Flux 类型的对象。下面我们给出使用范例，源码可以在文末获取。

## Dubbo RSocket 初体验

### 服务接口

```
1 public interface DemoService {  
2     Mono<String> requestMonoWithMonoArg(Mono<String> m1, Mono  
3     Flux<String> requestFluxWithFluxArg(Flux<String> f1, Flux  
4 }
```

```
1 <dependency>
2     <groupId>io.projectreactor</groupId>
3     <artifactId>reactor-core</artifactId>
4     <version>3.2.3-RELEASE</version>
5 </dependency>
```

在服务定义层，引入了 Mono，Flux 等 reactor 的概念，所以需要添加 reactor-core 的依赖。

## 服务提供者

```
1 public class DemoServiceImpl implements DemoService {
2     @Override
3     public Mono<String> requestMonoWithMonoArg(Mono<String> m1, Mono<String> m2) {
4         return m1.zipWith(m2, new BiFunction<String, String, String>() {
5             @Override
6             public String apply(String s, String s2) {
7                 return s+" "+s2;
8             }
9         });
10    }
11
12    @Override
13    public Flux<String> requestFluxWithFluxArg(Flux<String> f1, Flux<String> f2) {
14        return f1.zipWith(f2, new BiFunction<String, String, String>() {
15            @Override
16            public String apply(String s, String s2) {
17                return s+" "+s2;
18            }
19        });
20    }
21 }
```

除了常规的 Dubbo 必须依赖之外，还需要添加 dubbo-rsocket 的扩展

```
1 //... other dubbo moudle
```

```
2 <dependency>
3     <groupId>org.apache.dubbo</groupId>
4     <artifactId>dubbo-rpc-rsocket</artifactId>
5 </dependency>
```

配置并启动服务端，注意协议名字填写 rsocket：

```
1 <beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2       xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
3       xmlns="http://www.springframework.org/schema/beans"
4       xsi:schemaLocation="http://www.springframework.org/sc
5       http://dubbo.apache.org/schema/Dubbo http://dubbo.apa
6
7     <!-- provider's application name, used for tracing dependen
8     <dubbo:application name="demo-provider"/>
9
10    <!-- use registry center to export service -->
11    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
12
13    <!-- use Dubbo protocol to export service on port 20890
14    <dubbo:protocol name="rsocket" port="20890"/>
15
16    <!-- service implementation, as same as regular local bean
17    <bean id="demoService" class="org.apache.dubbo.samples.b
18
19    <!-- declare the service interface to be exported -->
20    <dubbo:service interface="org.apache.dubbo.samples.basic
21
22 </beans>
```

服务提供者的 bootstrap：

```
1 public class RsocketProvider {
2
3     public static void main(String[] args) throws Exception {
4         ClassPathXmlApplicationContext context = new ClassPat
```

```
5         context.start();
6         System.in.read(); // press any key to exit
7     }
8
9 }
```

## 服务消费者

然后配置并启动消费者消费者如下, 注意协议名填写 rsocket:

```
1  <beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2      xmlns:dubbo="http://dubbo.apache.org/schema/Dubbo"
3      xmlns="http://www.springframework.org/schema/beans"
4      xsi:schemaLocation="http://www.springframework.org/sc
5      http://dubbo.apache.org/schema/dubbo http://dubbo.apa
6
7      <!-- consumer's application name, used for tracing depen
8      don't set it same as provider -->
9      <dubbo:application name="demo-consumer"/>
10
11      <!-- use registry center to discover service -->
12      <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
13
14      <!-- generate proxy for the remote service, then demoSer
15      local regular interface -->
16      <dubbo:reference id="demoService" check="true" interface
17
18  </beans>
```

```
1  public class RsocketConsumer {
2
3      public static void main(String[] args) {
4          ClassPathXmlApplicationContext context = new ClassPa
5          context.start();
6          DemoService demoService = (DemoService) context.getB
```

```
7
8     while (true) {
9         try {
10             Mono<String> monoResult = demoService.reques
11             monoResult.doOnNext(new Consumer<String>() {
12                 @Override
13                 public void accept(String s) {
14                     System.out.println(s);
15                 }
16             }).block();
17
18             Flux<String> fluxResult = demoService.reques
19             fluxResult.doOnNext(new Consumer<String>() {
20                 @Override
21                 public void accept(String s) {
22                     System.out.println(s);
23                 }
24             }).blockLast();
25
26         } catch (Throwable throwable) {
27             throwable.printStackTrace();
28         }
29     }
30 }
31 }
```

可以看到配置上除了协议名使用 rsocket 以外其他并没有特殊之处。

## 实现原理

以前用户并不能在参数或者返回值里使用 Mono/Flux 这种流对象（reactive-stream里的流的概念）。因为流对象自带异步属性，当业务把流对象作为参数或者返回值传递给框架之后，框架并不能将流对象正确的进行序列化。

Dubbo 基于 RSocket 提供了 reactive 支持。RSocket 将 reactive 语义的复杂逻辑封装起来了，给上层提供了简洁的抽象如下：

```
1 Mono<Void> fireAndForget(Payload payload);
```

```
2
3 Mono<Payload> requestResponse(Payload payload);
4
5 Flux<Payload> requestStream(Payload payload);
6
7 Flux<Payload> requestChannel(Publisher<Payload> payloads);
```

- 从客户端视角看，框架建立连接之后，只需要将请求信息编码到 Payload 里，然后通过 requestStream 方法即可向服务端发起请求。
- 从服务端视角看，RSocket 收到请求之后，会调用我们实现的 requestStream 方法，我们从 Payload 里解码得到请求信息之后，调用业务方法，然后拿到 Flux 类型的返回值即可。

需要注意的是业务返回值一般是 `Flux<BizDO>`，而 RSocket 要求的是 `Flux<Payload>`，所以我们需要通过 map operator 拦截业务数据，将 BizDO 编码为 Payload 才可以递交给 RSocket。而 RSocket 会负责数据的传输和 reactive 语义的实现。

## 结语

Dubbo 2.7 相比 Dubbo 2.6 提供了 CompletableFuture 的异步化支持，在 Dubbo 3.0 又继续拥抱了 Reactive，不断对新特性的探索，无疑是增加了使用者的信心。RSocket 这一框架/协议，如今在国内外也是比较火的一个概念，它提供了丰富的 Reactive 语义以及多语言的支持，使得服务治理框架可以很快地借助它实现 Reactive 语义。有了响应式编程支持，业务可以更加方便的实现异步逻辑。

本篇文章对 Dubbo RSocket 进行了一个简单的介绍，对 Reactive、RSocket 感兴趣的同学也可以浏览下 Dubbo 3.0 源码对 RSocket 的封装。

相关链接：

[1] 文中源码：<https://github.com/apache/incubator-dubbo-samples/tree/3.x/dubbo-samples-rsocket>

[2] Dubbo 3.x 开发分支：<https://github.com/apache/incubator-Dubbo/tree/3.x-dev>

↪ 分享到

上一篇

研究网卡地址注册时的一点思考

下一篇

Dubbo2.7 三大新特性详解

Like

Issue Page

Error: Comments Not Initialized

Write

Preview

Login with GitHub

Leave a comment

Styling with Markdown is supported

Comment

Powered by Gitment

最新文章

技术杂谈

IDEA 插件推荐：CLOUD TOOLKIT 测评  
2019-06-27

RPC

研究网卡地址注册时的一点思考  
2019-04-29

RPC

【DUBBO3.0新特性】集成RSOCKET,新增响应式支持  
2019-04-11

RPC

DUBBO2.7 三大新特性详解

2019-03-21

数据库

一文探讨堆外内存的监控与回收

2019-03-17

## 分类

- [DevOps](#) (2)
- [Docker](#) (1)
- [JAVA](#) (22)
- [JAVA并发合集](#) (3)
- [JWT](#) (3)
- [Kong](#) (3)
- [MQ](#) (1)
- [RPC](#) (19)
- [Spring](#) (9)
- [Spring Cloud](#) (5)
- [Spring Data Redis](#) (2)
- [Spring Security](#) (7)
- [Spring Security OAuth2](#) (3)
- [Spring Session](#) (4)
- [响应式编程](#) (1)
- [技术杂谈](#) (15)
- [数据库](#) (4)
- [架构设计](#) (5)



- [规则引擎](#) (4)
- [领域驱动设计](#) (2)

## 归档

- [六月 2019](#) (1)
- [四月 2019](#) (2)
- [三月 2019](#) (4)
- [二月 2019](#) (1)
- [一月 2019](#) (4)
- [十二月 2018](#) (2)
- [十一月 2018](#) (1)
- [十月 2018](#) (1)
- [九月 2018](#) (4)
- [八月 2018](#) (3)
- [七月 2018](#) (3)
- [六月 2018](#) (1)
- [五月 2018](#) (4)
- [四月 2018](#) (9)
- [三月 2018](#) (3)
- [二月 2018](#) (3)
- [一月 2018](#) (3)
- [十二月 2017](#) (6)
- [十一月 2017](#) (5)

- [十月 2017](#) (6)
- [九月 2017](#) (12)
- [八月 2017](#) (10)
- [七月 2017](#) (1)
- [六月 2017](#) (2)
- [五月 2017](#) (1)
- [四月 2017](#) (6)
- [三月 2017](#) (4)
- [二月 2017](#) (7)
- [十一月 2016](#) (3)
- [八月 2016](#) (3)

## 标签云

Cloud Toolkit DUBBO DevOps DirectIO Docker Dubbo **JAVA** JCTools JMM JNA JWT Kong MQ  
Network PolarDB性能挑战赛 **RPC** Reactor RxJava Servlet **Spring** Spring Cloud Spring Cloud Zuul Spring  
Data Redis **Spring Security** Spring Security OAuth2 Spring Session TCP Validation XML Zipkin drools  
lua motan redis zookeeper 中文排版 事务 代码规范 多线程 开源 微服务 心跳 **技术杂谈** 数据库 文件IO  
杂谈 架构设计 求职 网关 网卡 规则引擎 队列 领域驱动设计

## 链接

- [namespace\\_ntyz](#)
- [程序猿DD|博客](#)
- [芋道源码](#)
- [匠心零度](#)
- [松花皮蛋的黑板报](#)

