Java 9 – 说说响应式流

Java amqp API

Harries Blog™ 2018-07-03 55 阅读

最初看到 Java 9 的这个新特性没太在意，及至重新关注到 Spring 5/Springboot 2 的 响应式 编程的时候才真正重视起 Reactive Streams(响应式流或反应式流)。应用响应式流的编程也就叫做响应式编程(Reactive Programming)，无论是 翻译 成反应式编程都有些令人摸不准头脑。与此对应的在 Web 设计方面有一个叫做响应式 Web 设计(Responsive web design)，两个词都译作响应式，却有些差别，大概是 Reactive 被译为反应的原因之一。

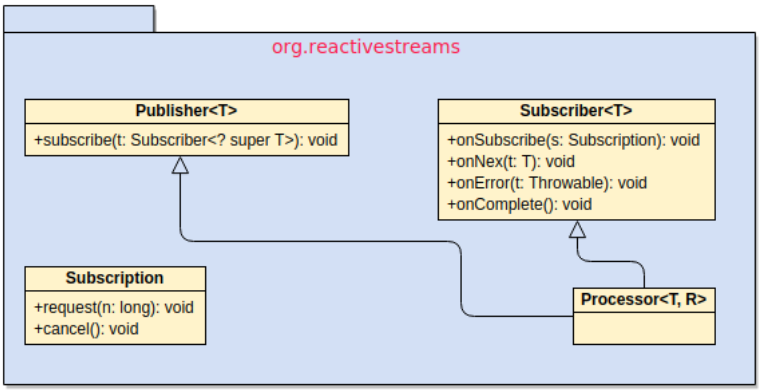
通过这里对 Reactive Streams 的学习，主要目的是为了进一步掌握 Spring 5/Springboot 2 的响应式 MVC 作铺垫的，不至于猛然间见 Flux, Mono 而不知所措。

函数式响应式编程 概念最早来自于九 十年 代末，这也激发了 微软 的 Erik Meijer 设计 开发 了 .NET 的 Rx(Reactive eXtension) 库，以及到后来 Netflix 的 RxJava 也与他有关系。Reactive Stream 更像是一种编程模式，致力于解决一个生产者产生一系列消息，一个或多个去消费它们的问题。两者的名词我们会用: producer - consumer (生产者-消费者), source/sink(水源/水槽, A kk a Stream 用了这个概念), publisher-subscriber(发布者-订阅者)。

既然 Reactive Stream 和 Java 8 引入的 Stream 都叫做流，它们之间有什么关系呢？有一点关系，Java 8 的 Stream 主要关注在流的过滤，映射，合并，而 Reactive Stream 更进一层，侧重的是流的产生与消费，即流在生产与消费者之间的协调。

一流的公司制定规范，Reactive Stream 标准在 2013 年也开始有了一个雏形：异步的流处理，并支持非阻塞式的 backpressure(背压? 很拗口的翻译，就是生产者与消费者之者应有流量控制)。流量控制即消费者的速率慢于生产者的速率时，生产者需要把速率降下来，比如说流处理时能在推/拉模式之间自动切换。至于背后的异步，非阻塞的实现仍然得仰仗于 多线程 了。

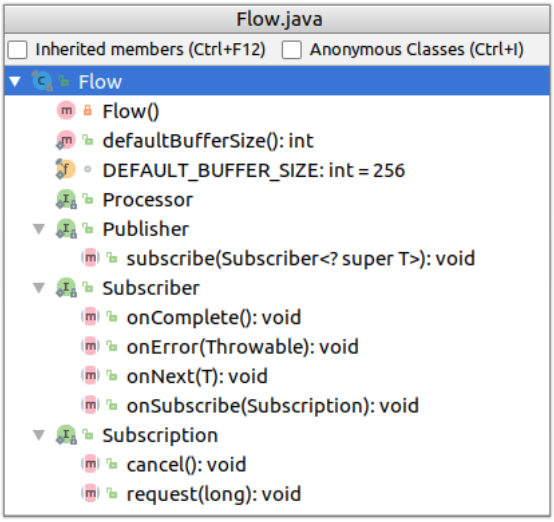
在之后出现了Netflix 的 RxJava, 它的四个主要角色是: `Observable` , `Observer` , `Subscriber` 和 `Subject` 。到 2015 年，正式的 Reactive Stream 出台，发布在 http ://www.reactive-streams.org/ 。从这里我们可以认识到规范有多粗暴，就是定义了四个接口，以及一句话说生产/消费者之间是异步的，并实现 backpresure，没有任何的实现参考。



在 Reactive Stream 规范正式出来后，RxJava 也向它靠拢，实现了 org.reactivestreams 中的以上四个接口，RxJava 2 更是重写了。见 Reactive Streams 1.0.0 is here , 看到该规范的拥趸还不少, 括号中为支持 Reactive Streams 的起始版本号。

- Akka Streams(1.0-RC2)
- MongoDB (1.0.0)
- Ratpack (0.9.16), 可用来创建非阻塞式 HTTP 应用
- Reactive Rabbit (1.0.0), Rabiit MQ /AMQP 的驱动
- Reactor (Spring 5 的响应式 MVC 就是用的它)
- RxJava (1.0.0), Netflix 出品
- Slick (3.0.0), Scala 的函数式关系映射组件，用于操作 数据库
- Vert.x 3.0 (milestone-5a), Ecl ip se 出品，也能用于构建非阻塞式 HTTP 应用

绕了一圈，该让 Java 9 与 Reactive Streams 发生关系了。Java 9 想必看到 Reactive Streams 是个好东西，于是把它纳入到 JDK 中来，但方式是无法容忍 JDK 中再出现 `org.reactivestreams` 这样的包定义，采用的做法是完全拷贝那四个接口定义，全收在了 `java.util.concurrent.Flow` 类中，作为 `Flow` 的内部静态接口存在。



JDK 9 本身也没有用力去实现以上四个接口，有两个比较简陋的 `SubmissionPublisher` 和 `ConsumerSubscriber` 。再就是还处于孵化器阶段的 `jdk.incubator.http` 包中的一些 `Publisher` ，`Subscriber` 实现，这也是 Reactive Streams 最应大力发挥网络 协议 领域。

因为有 JDK 9 的不遵循包名的引入 Reactive Streams 规范，所以 reactivestreams.org 又发出一个库 org.reactivestreams:reactive-streams-flow-adapters:1.0.2, 用于在 org.reactivestreams 和 java.util.consurrent.Flow 间的 Publisher, Subscriber, Processor, Subscription 之间的类型转换。

Spring 5 第一个 Release 版本在 2017-09-28 发布的，而 Java 9 是在 2017-07-27 正式发布的，就是说在 Spring 5 发布时已经有了 Java 的 Reactive Streams。不过 Spring 5 的第一个里程碑版还是在 2016-07-28, 所以那时选择了 Reactor , 所以要使用 Spring 5 的响应式编程就必须了解 Flux 和 Mono，或许下一个 Spring 版本也要适配 Java 9 的 Reactive Streams Flow API ，即双向转换或替换 API。

对于 Spring 5 可能发生的与 Java 9 Reactive Streams 的适配或许有些像 PlayFramework 兼容 Java 8 之前用的是它自己的 API `F.Option` 和 `F.Promise` ，后来从 Play 2.4 升级到 2.5 后完全采用了 Java 8 的 Optional 和 CompletionStage APIs 作为替代。

Java 9 的 SubmissionPublisher 应用 实例

前面提到过 Java 9 有两个简陋的 Publisher 和 Subscriber 实现，来看看 `SubmissionPublisher` 和 `ConsumerSubscriber` 的应用举例

```
package cc.unmi;

import java.util.concurrent.CompletableFuture;
import java.util.concurrent.SubmissionPublisher;
import java.util.stream.IntStream;

public class TestFlow {

    public static void main(String[] args) {
        CompletableFuture<Void> subTask;
        try (SubmissionPublisher<Integer> publisher = new SubmissionPublisher<>()) {
            subTask = publisher.consume(System.out::println);
            IntStream.rangeClosed(1, 3).forEach(publisher::submit);
        }

        subTask.join();
    }
}
```

运行，输出为

1

2

3

没什么意外，看起来和直接用 Stream API 差不多，效果与下面仅一行 代码 是一样的

```
IntStream.rangeClosed(1, 3).forEach(System.out::println);
```

面实际上内部运作起来就完全是另一回事了，此间就有 Java Flow APIs 在运转。下面逐步来理解一下：

从 SubmissionPublisher 构造函数起

SubmissionPublisher 实现了 Flow.Publisher 接口，它有三个构造函数

SubmissionPublisher() //默认 线程池 为 ForkJoinPool.commonPool(), 缓冲区大小为 256

SubmissionPublisher(Executor executor , int maxBufferCapacity)

SubmissionPublisher(Executor executor, int maxBufferCapacity, BiConsumer<? super Flow.Subscriber<? super T>, ? super Throwable> handler)

publisher.consume(consumer) 发生了什么

看 SubmissionPublisher 的 consumer(…) 方法

```
    public CompletableFuture<Void> consume(Consumer<? super T> consumer) {
        if (consumer == null)
            throw new NullPointerException();
        CompletableFuture<Void> status = new CompletableFuture<>();
        subscribe(new ConsumerSubscriber<T>(status, consumer));
        return status;
    }
```

上面代码创建了一个 `ConsumerSubscriber` 实例，它实现了 Flow.Subscriber 接口，subscribe(…) 方法创建了 Subscription 实例

```
BufferedSubscription<T> subscription = new BufferedSubscription<T>(subscriber, executor, onNextHandler, maxBufferCapacity);
```

并提交任务给 线程 池，该任务执行到了 ConsumerSubscriber 的 onSubscribe(Flow.Subscription subscription) 方法，看到

```
subscription.request(Long.MAX_VALUE);
```

一下请求所有的元素。

publisher.submit(T item) 生产消息

SubmissionPublisher.submit(item) 发布消息后，ConsumerSubscriber 会收到 onNext, onComplete 事件，或出错时的 onError，对应方法

```
onNext(T item)
void onComplete()
void onError(Throwable ex)
```

从上面大概能看到一个 Reactive Streams 应用有 Publisher, Subscriber, Subscription 多个角色在参与协作。而一个 Reactive Streams 组件要做的事情就是就是尽可能的把它们做的更完美，高效率且接口更友好。

SubmissionPublisher 可有多个订阅者

当给 SubmissionPublisher 指定多个 Subscriber 的时候，消息只需发布一次，这与 `IntStream.rangeClosed(1, 3).forEach(System.out::println);` 就不一样了

```
    public static void main(String[] args) {
        CompletableFuture<Void> subTask1;
        CompletableFuture<Void> subTask2;
        try (SubmissionPublisher<Integer> publisher = new SubmissionPublisher<>()) {
            subTask1 = publisher.consume(System.out::print);
            subTask2 = publisher.consume(System.out::print);
            IntStream.rangeClosed(1, 3).forEach(publisher::submit);
        }

        subTask1.join();
        subTask2.join();
    }
```

执行后的输出顺序是不确定的，可能是下面任意情况

Reactive Streams 和 Actor

在进行异步消息处理时，Reactive Streams 和 Actor 是两种不同的编程模式选择。Reactive Streams 规范相比 Actor 更简单，只是说收发消息异步，有流量控制。而 Actor 编程模式涉及到 Actor 容错 管理 ，消息路由， 集群 ，并支持远程消息等。

还有共同之处是: 它们定义的 API 都很简单，编码时都基本不需要关注线程本身，而实际消息的传递都是背后的线程池。所以线程的 配置 可延迟到部署阶段来进行优化处理。

下一步，继续对 Spring 5 的响应式 MVC 应用进行实战体验

*原文*https ://yanbin.blog/java-9-talk-reactive-stream/

转载请注明原文出处： Harries Blog™ » Java 9 – 说说响应式流

Java amqp API

点赞

作者：Harries Blog™

追心中的海，逐世界的梦

原文地址：Java 9 – 说说响应式流, 感谢原作者分享。

←微信支付JAVA SDK存在漏洞，可导致商家服务器被入侵（绕过支付）

→Java 9 – 说说响应式流

发表评论

发表评论

您可能感兴趣的博客

您可能感兴趣的代码