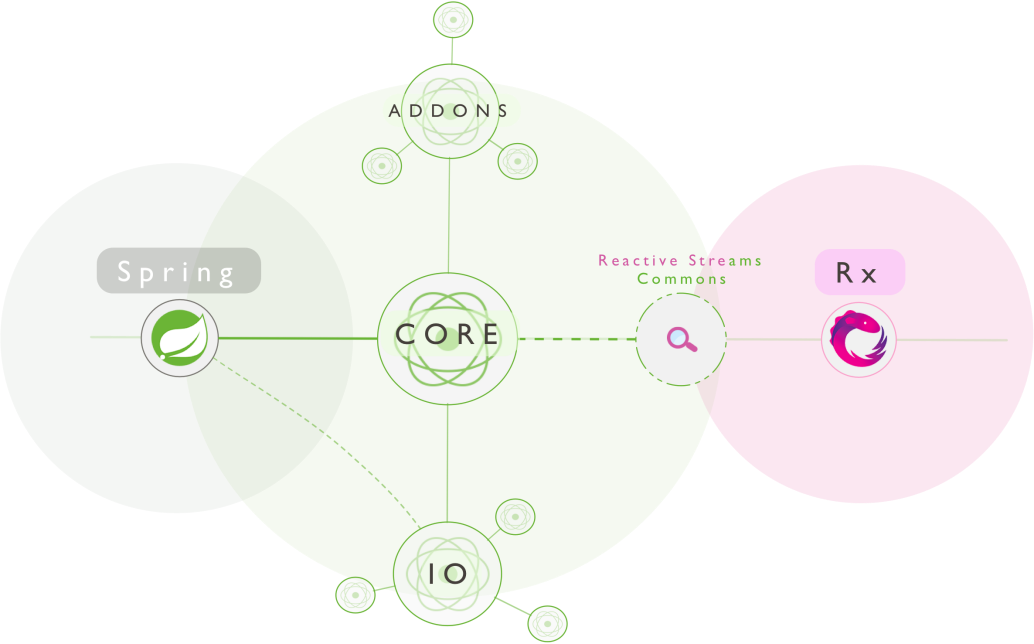
# Project Reactor

响应式编程是一种编程风格，提供基于事件驱动的方式来处理异步数据流的能力，其遵循观察者设计模式，当事物发生状态变化时，其他事物将被相应地通知和更新，观察者不需要轮询事件的变化，而是异步等待事件变化的通知，观察者收到通知后处理该事件。响应式编程是基于Reactor思想，目前比较流行的是RxJava，另一种常用的响应式编程库Project Reactor，Spring 5使用Reactor库作为编程基础，其相关类库包括：



* Reactor Core，响应式编程基础，核心是Mono及Flux，其提供丰富的响应式API扩展
* Reactor IO，响应式编程的网络基础库，包括reactor ipc,reactor-codec及reactor-netty等
* Reactor Addons，响应式插件的插件，包括reactor-adpater,reactor-logback等
* 其他扩展库，reactor-amqp、reactor-kafka、reactor-bus等
* Reactor-Spring，在Spring5中包括Spring-WebFlux

# Reactor-Core

添加Reactor依赖，如下所示：

*<dependency>*

*<groupId>io.projectreactor</groupId>*

*<artifactId>reactor-core</artifactId>*

*<version>3.2.3.RELEASE</version>*

*</dependency>*

在Reactor中核心概念是Flux及Mono

1. **Flux使用示例**

*Flux.just("Hello", "World").subscribe(System.out::println);*

*Flux.fromArray(new Integer[] {1, 2, 3}).subscribe(System.out::println);*

*Flux.empty().subscribe(System.out::println);*

*Flux.range(1, 10).subscribe(System.out::println);*

*Flux.interval(Duration.of(10, ChronoUnit.SECONDS)).subscribe(System.out::println);*

Flux类似于RxJava的Observable，触发零到多个事件，并根据实际情况处理或者触发错误。

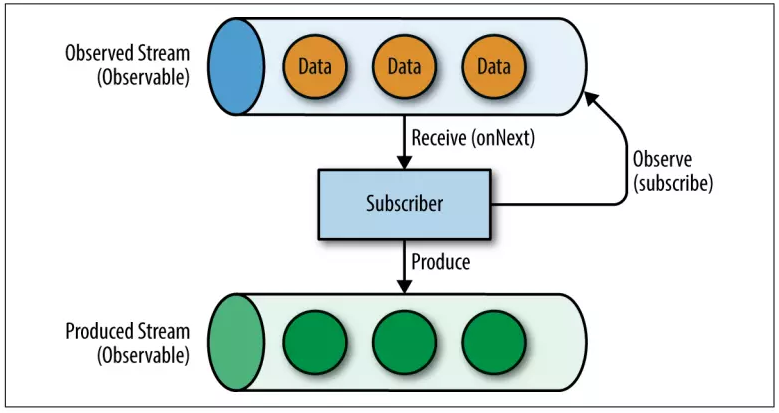
1. **Mono的使用示例**

*Mono.fromSupplier(() -> "Hello").subscribe(System.out::println); Mono.justOrEmpty(Optional.of("Hello")).subscribe(System.out::println); Mono.create(sink -> sink.success("Hello")).subscribe(System.out::println)*

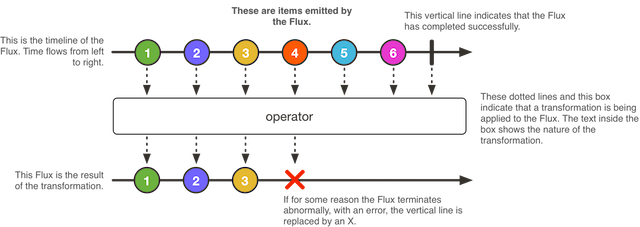
Mono最多触发一次事件，其对应RxJAVA的Single及Maybe类型。

1. **Publisher**

Reactive编程的开发模型，其专注于数据流向、对变化的反馈以及传播它们，如下图所示：



Mono及Flux都是实现Publisher接口的抽象类，Flux是标准的Publisher，产生响应式流，如下图所示：



Flux的核心定义如下：

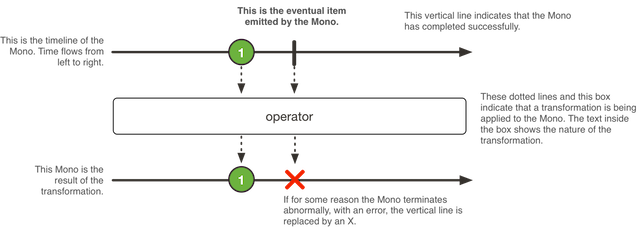
*public abstract class Flux<T> implements Publisher<T>{*

*public final Mono<T> doOnNext()；*

*public final Flux<T> doOnComplete(Runnable onComplete);*

*public final Flux<T> onErrorX(); ......}*

另外Flux.interval，生成Flux<Long>流，用于实现时钟流转。Mono是特殊的Publisher，生成至多一个数据，如下图：



# Reactor Netty

基于Netty框架实现响应式TCP/HTTP的客户端及Server，其使用添加依赖如下：

*<dependency>*

*<groupId>io.projectreactor.netty</groupId>*

*<artifactId>reactor-netty</artifactId>*

*<version>0.8.3.RELEASE</version>*

*</dependency>*

1. 创建Netty TcpServer，代码如下：

*DisposableServer server = TcpServer.create()*

*.port(0)*

*.handle((in, out) -> {*

*in.receive()*

*.log("channel")*

*.subscribe(trip -> latch.countDown());*

*return Flux.never();*

*})*

*.wiretap(true)*

*.bindNow();*

1. 创建Netty TcpClient，代码如下：

*Connection client = TcpClient.create()*

*.port(server.address()*

*.getPort())*

*.handle((in, out) -> out.sendString(Flux.just("test")))*

*.wiretap(true)*

*.connectNow();*

1. 源码分析，Netty TcpServer的核心是TcpServerHandle，其定义如下：

*final class TcpServerHandle extends TcpServerOperator implements ConnectionObserver {*

*final BiFunction<? super NettyInbound,*

*? super NettyOutbound,*

*? extends Publisher<Void>> handler;*

*}*

具体使用见TcpServer.handle方法，TcpClient也是通过TcpServerHandle处理。核心类：

* NettyInbound，接收信息，
* NettyOutbound，发送消息类，核心方法：

*default NettyOutbound send(Publisher<? extends ByteBuf> dataStream) {  
 return sendObject(dataStream);  
 }*

将数据发送到客户端。

https://violetagg.github.io/reactor-netty-workshop/

https://www.jianshu.com/p/42442b47900b

https://github.com/reactor/reactor-netty

# Spring Reactive

Spring 5.0提供响应式编程，其核心模块是spring-web-reactive和spring-webflux，其基于Reactor Netty模块提供网络通信。Spring WebFlux模块中包含响应式HTTP、服务器推送事件和Web客户端/服务器端的支持。创建Spring Boot项目，添加相关依赖：

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-webflux</artifactId>*

*</dependency>*

WebFlux支持三种形式的访问：REST API、服务器推送事件及WebSocket

1. REST API
2. SSE，服务器推送事件
3. WebSocket

具体见https://www.ibm.com/developerworks/cn/java/spring5-webflux-reactive/index.html

# Vert.x Reactive

在Vert.x中也提供Reactive编程，但是其基于RxJava类库，这里简单介绍其使用

1）添加依赖

在Vert.x中提供了RxJava的支持，添加依赖如下：

*<dependency>*

*<groupId>io.vertx</groupId>*

*<artifactId>vertx-core</artifactId>*

*<version>${vertx.version}</version>*

*</dependency>*

*<dependency>*

*<groupId>io.vertx</groupId>*

*<artifactId>vertx-hazelcast</artifactId>*

*<version>${vertx.version}</version>*

*</dependency>*

*<dependency>*

*<groupId>io.vertx</groupId>*

*<artifactId>vertx-rx-java</artifactId>*

*<version>${vertx.version}</version>*

*</dependency>*

2） 编写Receiver类

*public class Sender extends AbstractVerticle {*

*@Override*

*public void start() throws Exception {*

*EventBus eb = vertx.eventBus();*

*vertx.setPeriodic(1000,*

*v -> eb.publish("news-feed", "Some news!"));*

*}*

*}*

3）编写Sender类

*public class Receiver extends AbstractVerticle {*

*@Override*

*public void start() throws Exception {*

*EventBus eb = vertx.eventBus();*

*eb.consumer("news-feed")*

*.toObservable()*

*.subscribe( message ->*

*System.out.println("Received news: " + message.body()));*

*System.out.println("Ready!");*

*}*

*}*

4）测试

* 运行Reciver，其输出如下：

*Ready!*

* 运行Sender，在Reciver端输出如下：

*Recycler - -Dio.netty.recycler.ratio: 8*

*Received news: Some news!*

*Received news: Some news!*

https://projectreactor.io/docs/core/release/reference/#flux

https://www.jianshu.com/p/5172c48cb877

https://www.infoq.com/articles/reactor-by-example

https://en.wikipedia.org/wiki/Reactive\_programming

https://projectreactor.io/

https://github.com/reactor

https://www.infoq.com/articles/rxjava-by-example