6月29日（周三）
晚8点-9点

SpringBoot3升级指南

主讲内容：

* SpringBoot 版本更替说明
* 如何升级到SpringBoot3
* 响应式编程学习路径
* SpringBoot3.x 特性简介
* 响应式编程与Servlet对比

直播间福利大礼：

《SpringBoot升级指南.ppt》
《响应式编程入门.pdf》
《响应式编程进阶.pdf》
《SpringBoot3 最新知识大纲.xmind》

雷丰阳
江湖人称“雷神”
技术大神，助力谷粉打开大厂之门

直播资料获取

公开课3群：621404992
公开课4群：376909204

课程咨询添加

柳儿姐vx：
w13021299785

小谷姐姐vx：
atguigubj

20点准时开讲！

SpringBoot3升级指南

讲师：雷丰阳

目录

1 Spring版本更替

2 SpringBoot3快速浏览

3 理解响应式编程

4 响应式编程示例

尚硅谷

1 Spring版本更替

Spring版本重要特性变化

2003

2004/3/24
Spring1.0
IOC、AOP、XML

Spring3.x
2009/12/16

注解驱动

利用java5注解，完全改进
...

Spring4.x
2013/11/1

支持JAVA8

Reactor、架构重构
....

Spring5.x
2017/9/28

响应式编程

Spring WebFlux、junit5、
Kotlin ..

Spring6.x
2021/9/1

Java17全面升级

Jakarta EE 9、AOT、
云原生方向、GraalVM

SpringBoot迭代过程

微服务

提供微服务开发基本能力

快速一站式整合、Spring4

SpringBoot 1.0
2014年1月22

SpringBoot 2.0.0
2018年3月1

响应式编程

Spring5、Spring WebFlux

Java17

Spring6、容器镜像、AOT

CloudNative

SpringBoot3.0
2022 年 1 月 20

2 SpringBoot3.0快速浏览

SpringBoot3
概览.xmind

Spring6.0核心
.xmind

3 理解响应式编程

什么是响应式编程


响应式编程
网页
图片
视频
学术
词典
地图
506,000,000 条结果
时间不限
面向数据流和变化传播的声明式编程范式
根据 2 个来源
在计算中，响应式编程是一种面向数据流和变化传播的声明式编程范式。
在计算中，响应式编程或反应式编程是一种面向数据流和变化传播的声明式编程范式。

响应式

异步

响应式

高效消息传递

非阻塞IO

序列流

流水线操作&背压

响应式

- https://www.reactivemanifesto.org/zh-CN
- https://github.com/reactive-streams/reactive-streams-jvm

场景：服务员（线程） - 顾客点餐（请求）

非阻塞IO

阻塞式IO：
1、等待上一个顾客点餐结束，服务员去服务下一个顾客
2、或者招大量服务员，同时服务多个顾客

非阻塞式IO：
1、菜单给上一个顾客，菜单给下一个顾客，谁点完了叫服务员（回调）。
2、菜单给上一个顾客，菜单给下一个顾客，问上一个顾客是否点完，问下一个顾客是否点完（Future）

共识：阻塞是浪费

```
public static void blocking() {
    StopWatch watch = new StopWatch();
    watch.start();
    String baseInfo = restTemplate.getForObject(URI.create("http://localhost:8000/baseinfo"), String.class);
    BigDecimal price = restTemplate.getForObject(URI.create("http://localhost:8000/price"), BigDecimal.class);
    String coupon = restTemplate.getForObject(URI.create("http://localhost:8000/coupon"), String.class);
    String recommend = restTemplate.getForObject(URI.create("http://localhost:8000/recommend"), String.class);
    String stock = restTemplate.getForObject(URI.create("http://localhost:8000/stock"), String.class);
    ProductDetail productDetail = new ProductDetail(baseInfo, price, coupon, recommend, stock);
    watch.stop();
    System.out.printf("【商品信息】 [%s]  耗时：[%s]ms",productDetail,watch.getTotalTimeMillis());
}
```

非阻塞

Callbacks机制 → ApplicationListener EventListener

Future异步 → 立即返回Future<T> get()获取结果

```
public static void noblocking() throws ExecutionException, InterruptedException {
    StopWatch watch = new StopWatch();
    watch.start();
    CompletableFuture<String> baseInfo = CompletableFuture.supplyAsync(() -> restTemplate.getForObject(URI.create("http://localhost:8000/baseinfo"), String.class));
    CompletableFuture<BigDecimal> price = CompletableFuture.supplyAsync(() -> restTemplate.getForObject(URI.create("http://localhost:8000/price"), BigDecimal.class));
    CompletableFuture<String> coupon = CompletableFuture.supplyAsync(() -> restTemplate.getForObject(URI.create("http://localhost:8000/coupon"), String.class));
    CompletableFuture<String> recommend = CompletableFuture.supplyAsync(() -> restTemplate.getForObject(URI.create("http://localhost:8000/recommend"), String.class));
    CompletableFuture<String> stock = CompletableFuture.supplyAsync(() -> restTemplate.getForObject(URI.create("http://localhost:8000/stock"), String.class));
    ProductDetail productDetail = new ProductDetail(baseInfo.get(), price.get(), coupon.get(), recommend.get(), stock.get());
    watch.stop();
    System.out.printf("【商品信息】 [%s]  耗时：[%s]ms",productDetail,watch.getTotalTimeMillis());
}
```

万物皆数据

中间操作　　中间操作

Publisher ⟷ **Subscriber**

Subscription

响应式数据处理，类似装配线一样。

原始数据从 **Publisher** 流入装配线使用各种 **operator** 加工后推给**Subscriber**

Mono[0|1]　　Flux[N]

- 如何定义流水线?
- 原料如何流入?
- 如何让流水线上的工人将处理过的原料交给下一个工人?
- 流水线何时开始运行?
- 流水线何时结束运行?

发布者与订阅者

```
//1、创建一个发布者
Flow.Publisher publisher = new SubmissionPublisher();

//2、创建一个订阅者
Flow.Subscriber subscriber = new Flow.Subscriber() {...};

//3、建立关系
publisher.subscribe(subscriber);

//4、发送数据测试
((SubmissionPublisher)publisher).submit( item: "a");
```

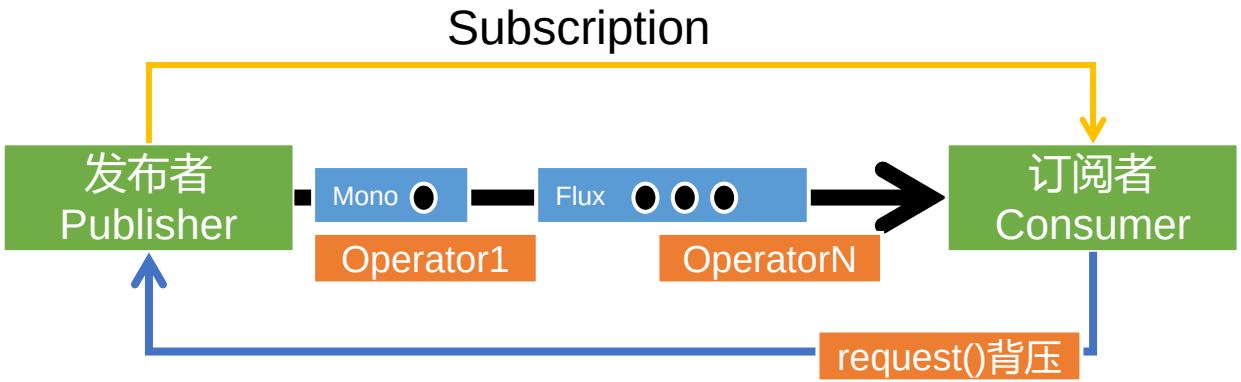
```java
//2、创建一个订阅者
Flow.Subscriber subscriber = new Flow.Subscriber() {
    Flow.Subscription subscription; //订阅契约
    @Override
    public void onSubscribe(Flow.Subscription subscription) {
        System.out.println("建立订阅契约");
        this.subscription = subscription;
        subscription.request( n: 1);//请求上游一个数据
    }

    @Override
    public void onNext(Object item) {
        System.out.println("订阅者收到数据："+item);
        //业务处理
        subscription.request( n: 1);//请求上游一个数据。背压
    }

    @Override
    public void onError(Throwable throwable) {
        System.out.println("发生异常："+throwable);
        subscription.cancel(); //取消订阅
    }

    @Override
    public void onComplete() {
        System.out.println("数据接受完成");
    }
};
```

中间处理器

```
class MyProcessor extends SubmissionPublisher<String> implements Flow.Processor<String,String>{
    Flow.Subscription subscription;
    @Override
    public void onSubscribe(Flow.Subscription subscription) {
        System.out.println("中间MyProcessor建立订阅关系");
        subscription.request( n: 1);//找到上游要数据
    }

    @Override
    public void onNext(String item) {
        //数据再加工
        String val = item + "-x";
        //数据发出去
        this.submit(val);
        subscription.request( n: 1);//找到上游要数据
    }

    @Override
    public void onError(Throwable throwable) {

    }

    @Override
    public void onComplete() {

    }
}
```

```
public static void main(String[] args) throws InterruptedException {
    MyProcessor myProcessor = new MyProcessor();
    //1、创建一个发布者
    Flow.Publisher publisher = new SubmissionPublisher();

    //2、创建一个订阅者
    Flow.Subscriber subscriber = new Flow.Subscriber() {...};

    //3、发布者-处理器 建立订阅关系
    publisher.subscribe(myProcessor);

    //4、处理器-订阅者 建立订阅关系
    myProcessor.subscribe(subscriber);

    //4、发送数据测试
    ((SubmissionPublisher)publisher).submit( item: "a");
      ((SubmissionPublisher)publisher).close();

    Thread.currentThread().join();
}
```

响应式



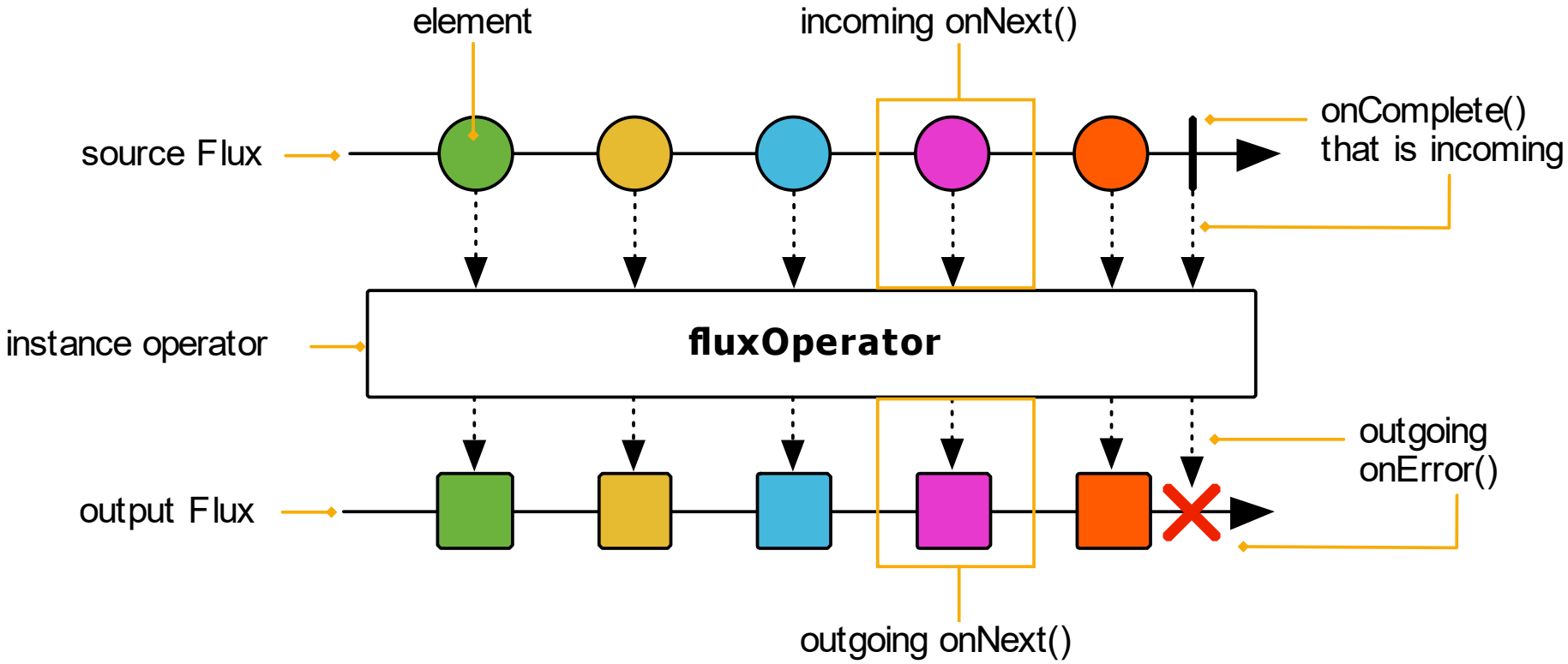

```
LongAdder longAdder = new LongAdder();
Flux<Long> flux = Flux.generate(longSynchronousSink -> {
    System.out.println("生产数据："+longAdder);
    longSynchronousSink.next(longAdder.longValue());
    longAdder.increment();
});

//1、不订阅、无效果
flux.subscribe(item->{
    System.out.println(item);
});
```

直到subscribe，
否则不会发生任何事情

背压

```
flux.subscribe(new BaseSubscriber<Long>() {
    @SneakyThrows
    @Override
    protected void hookOnNext(Long value) {
        System.out.println("==>"+value);
          TimeUnit.SECONDS.sleep(3);
        request( n: 1);
    }

    @Override
    protected void hookOnSubscribe(Subscription subscription) {
        request( n: 1);
    }
});
```

操作符


element
incoming onNext()
source Flux
onComplete()
that is incoming
instance operator
fluxOperator
outgoing
onError()
output Flux
outgoing onNext()

高效消息传递

Reactor：每秒1亿的消息速率，却能占用更少的内存和线程资源

从命令式到响应式编程

- Reactive 库（例如Reactor）旨在解决JVM上“经典”异步方法的这些缺点，同时还要关注其他一些方面：
- - 组合性和可读性（Composability and Readability）
- - 使用丰富的 操作符，操作管理一个数据流
- - 不 subscribe，就不会有事情发生
- - 背压 或 消费者通过给生产者信号来控制数据的生产速率不会太高
- - 高阶高度抽象

误区一：响应式更快

- 响应式并不会更快，相反由于维护数据流等可能会比异步慢一些
- web的响应式旨在用少量的线程干更多的事情。而不是每个请求一个线程

误区二：响应式难

- 响应式就几个核心概念
  - Publisher 发布者
  - Subscriber 订阅者
  - Sink 水槽
  - Mono 0-1 个数据
  - Flux N 个数据
  - Operators 操作符

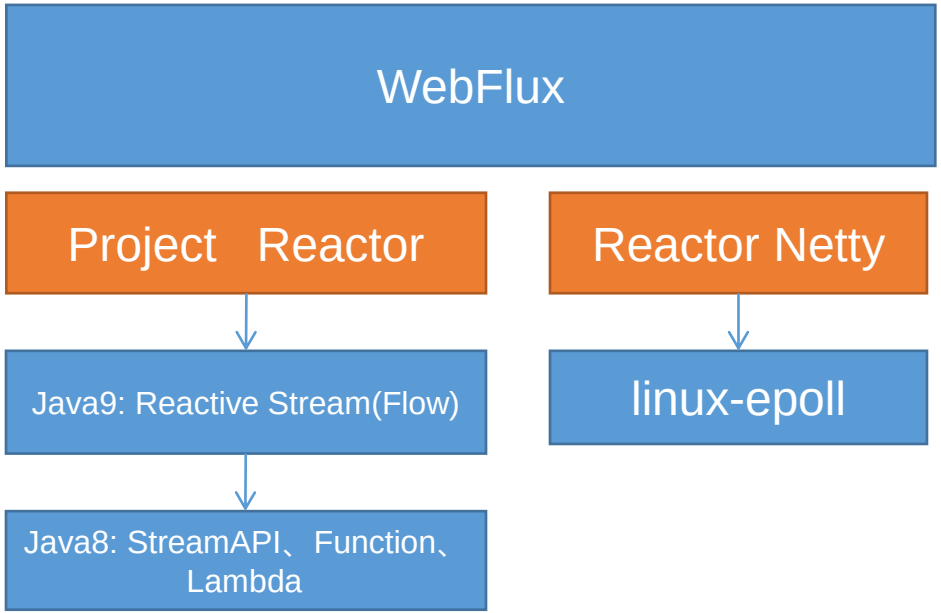
- API不难，难的是适应新的编程模式，改变旧的思维方式

StreamAPI　Lambda　Function　Reactor

误区三：响应式不能应用所有场景

- 万物皆可响应式
- 因为响应式可以理解为：
  - 多线程
  - 异步
  - 消息队列
  - 发布订阅感知

理解关系


WebFlux
Project Reactor
Reactor Netty
Java9: Reactive Stream(Flow)
linux-epoll
Java8: StreamAPI、Function、Lambda

4 响应式编程示例

WebFlux

响应式与Servlet对比-常用API

API功能	Servlet	WebFlux
前端控制器	DispatcherServlet	DispatcherHandler
处理器	Controller	WebHandler
请求、响应	ServletRequest、ServletResponse	ServerWebExchange
过滤器	Filter（HttpFilter）	WebFilter
异常处理器	HandlerExceptionResolver	WebExceptionHandler
web配置	@EnableWebMvc	@EnableWebFlux
自定义配置	WebMvcConfigurer	WebFluxConfigurer
返回结果	任意	Mono、Flux
发送REST请求	RestTemplate	WebClient

谢谢观看

6月30日（周四）
晚8点-9点

搞定90%+互联网项目高并发问题

主讲内容：

* QPS、TPS数据指标
* 系统压测方案及测试工具
* 全栈高并发系统技术架构演化
* 多级缓存设计及落地
* 调优：Sendfile数据“0拷贝”内部执行过程
* 互动答疑

直播间福利大礼：

《高并发系统架构演化图》
《Nginx高并发网站技术架构实战.pdf》

张一明
资深架构师，行业老司机
先后担任多家公司的技术总监

直播资料获取

公开课3群：621404992
公开课4群：376909204

课程咨询添加

柳儿姐vx：
w13021299785

小谷姐姐vx：
atguigubj