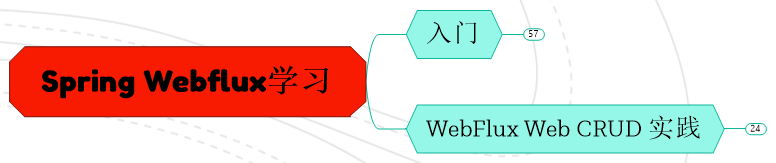
Spring Webflux学习



1. 入门 3

1.1. 什么是 Spring WebFlux？ 3

1.1.1. Spring Webflux是一个**非阻塞的web框架**。它**能够充分利用多核 CPU 的硬件资源去处理大量的并发请求。** 3

1.2. 为什么使用Spring Webflux？ 3

1.2.1. 优点 3

· **在不扩充硬件资源的前提下，提升系统的吞吐量和伸缩性。** 3

1.2.2. 缺点 3

· **WebFlux 并不能使接口的响应时间缩短，它仅仅能够提升吞吐量和伸缩性。** 4

1.3. Webflux有什么使用场景？ 4

1.3.1. Webflux特别适合应用在 IO 密集型的服务中，比如微服务网关这样的应用中。 4

1.4. 选 WebFlux 还是 Spring MVC? 4

1.4.1. **WebFlux 不是 Spring MVC 的替代方案**！（目前是这样） 4

1.4.2. Spring MVC一个请求一个线程不合适IO密集型服务 4

· Spring MVC 构建于 Servlet API 之上，使用的是同步阻塞式 I/O 模型，什么是同步阻塞式 I/O 模型呢？就是说，每一个请求对应一个线程去处理。所以spring mvc就已经定死了，**一个请求一条线程，可以得出SpringMVC不合适多IO操作。** 4

· 而SpringMVC的缺点恰恰被Spring Webflux所填补， Spring Webflux合适多IO场景 4

· 如果你目前在 Spring MVC 框架中大量使用非同步方案，那么，WebFlux 才是你想要的，否则，使用 Spring MVC 才是你的首选。 4

1.4.3. Spring MVC 因为是使用的同步阻塞式，更方便开发人员编写功能代码，Debug 测试等，一般来说，如果 Spring MVC 能够满足的场景，就尽量不要用 WebFlux; 4

1.5. 异同点 5

1.5.1. 5

· 相同点 5

· 都可以使用 Spring MVC 注解，如 @Controller, 方便我们在两个 Web 框架中自由转换； 5

· 均可以使用 Tomcat, Jetty, Undertow Servlet 容器（Servlet 3.1+）； 5

· 注意点 5

· Spring MVC 因为是使用的同步阻塞式，更方便开发人员编写功能代码，Debug 测试等，一般来说，如果 Spring MVC 能够满足的场景，就尽量不要用 WebFlux; 5

· WebFlux 默认情况下使用 Netty 作为服务器; 5

· WebFlux 目前不支持 MySql（只是生产上不支持，但实际上支持） 5

· 你可能会说可以用Spring官网的R2DBC，但请注意，R2DBC虽然支持mysql，但该项目已经很久没有维护了 6

· 6

· 6

· 况且，你需要注意，该项目是由个人开发，本身就存在风险 6

· 6

· 6

1.6. 简单看看 WebFlux 是如何分发请求的 7

1.6.1. Spring MVC 的前端控制器是 DispatcherServlet, 而 WebFlux 是 DispatcherHandler，它实现了 WebHandler 接口 7

· WebHandler 接口 7

· 查看DispatcherHandler类中处理请求的 handle 方法 7

· ①：ServerWebExchange 对象中放置每一次 HTTP 请求响应信息，包括参数等； 7

· ②： 判断整个接口映射 mappings 集合是否为空，空则创建一个 Not Found 的错误； 8

· ③： 根据具体的请求地址获取对应的 handlerMapping; 8

· ④： 调用具体业务方法，也就是我们定义的接口方法； 8

· ⑤： 处理返回的结果； 8

1.7. 快速入门 8

1.7.1. 全局Router API方式 8

· 添加依赖 8

· 启动类 8

· 添加Handler 9

· 添加Router 9

· 启动启动类，访问http://localhost:8080/hello 9

· 9

1.7.2. 使用SpringMVC那套注解 9

· 添加依赖 9

· 添加启动类 10

· 添加Controller 10

· 启动项目，访问http://localhost:8080/user 10

· Webflux的server-sent event 11

· 这样做会告诉客户端，我服务端的请求是一个流，请客户端不要关闭请求链接，等待下一个包 11

· 编写测试类 11

· 需要额外导入 11

2. WebFlux Web CRUD 实践 12

2.1. mysql crud 12

2.1.1. 思路 12

· Spring WebFlux 使用 R2DBC 访问 MySQL 12

· https://github.com/mirromutth/r2dbc-mysql这个项目很久没有更新了，找了个新的non-blocking驱动 12

· https://github.com/jasync-sql/jasync-sql 12

2.1.2. 步骤 12

· 导入依赖 12

· application.yml 13

· 启动类 13

· 配置类 13

· 没有这个配置类就会报错 14

· 我们可以根据错误的调用栈找到这里 15

· 15

· 根据前面我们学过的@Bean必须要特化，所以这里是ConnectionPool 16

· 但实际上它是 16

· 16

· 从这里我们也能判断的出来 16

· 16

· 根据我们找到的Bean配置，知道需要配置一个属于自己的ConnectionFactory或者ConnectionPool 16

· 16

· domain 17

· repository 17

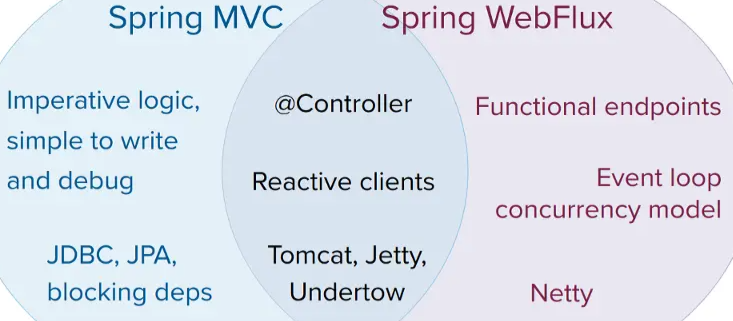
· controller 17

2.1.3. 这样我们就能完成Spring Webflux访问mysql的功能了 18

1. 入门
   1. 什么是 Spring WebFlux？
      1. Spring Webflux是一个**非阻塞的web框架**。它**能够充分利用多核 CPU 的硬件资源去处理大量的并发请求。**
   2. 为什么使用Spring Webflux？
      1. 优点
         * **在不扩充硬件资源的前提下，提升系统的吞吐量和伸缩性。**
      2. 缺点
         * **WebFlux 并不能使接口的响应时间缩短，它仅仅能够提升吞吐量和伸缩性。**
   3. Webflux有什么使用场景？
      1. Webflux特别适合应用在 IO 密集型的服务中，比如微服务网关这样的应用中。

PS: IO 密集型包括：**磁盘IO密集型, 网络IO密集型**，微服务网关就属于网络 IO 密集型，使用异步非阻塞式编程模型，能够显著地提升网关对下游服务转发的吞吐量。

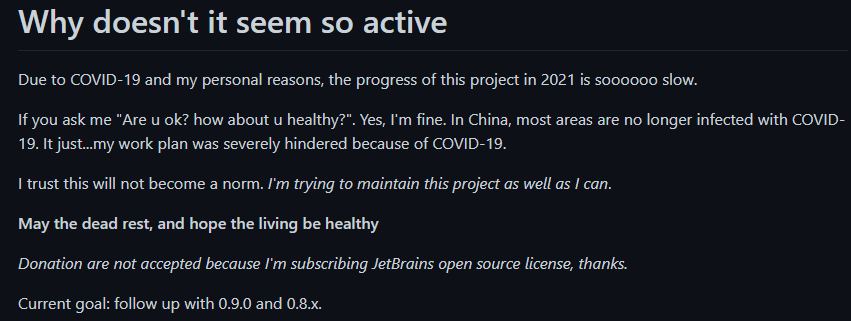
* 1. 选 WebFlux 还是 Spring MVC?
     1. **WebFlux 不是 Spring MVC 的替代方案**！（目前是这样）
     2. Spring MVC一个请求一个线程不合适IO密集型服务
        + Spring MVC 构建于 Servlet API 之上，使用的是同步阻塞式 I/O 模型，什么是同步阻塞式 I/O 模型呢？就是说，每一个请求对应一个线程去处理。所以spring mvc就已经定死了，**一个请求一条线程，可以得出SpringMVC不合适多IO操作。**
        + 而SpringMVC的缺点恰恰被Spring Webflux所填补， Spring Webflux合适多IO场景
        + 如果你目前在 Spring MVC 框架中大量使用非同步方案，那么，WebFlux 才是你想要的，否则，使用 Spring MVC 才是你的首选。
     3. Spring MVC 因为是使用的同步阻塞式，更方便开发人员编写功能代码，Debug 测试等，一般来说，如果 Spring MVC 能够满足的场景，就尽量不要用 WebFlux;
  2. 异同点



* + - * 相同点
        + 都可以使用 Spring MVC 注解，如 @Controller, 方便我们在两个 Web 框架中自由转换；
        + 均可以使用 Tomcat, Jetty, Undertow Servlet 容器（Servlet 3.1+）；
      * 注意点
        + Spring MVC 因为是使用的同步阻塞式，更方便开发人员编写功能代码，Debug 测试等，一般来说，如果 Spring MVC 能够满足的场景，就尽量不要用 WebFlux;
        + WebFlux 默认情况下使用 Netty 作为服务器;
        + WebFlux 目前不支持 MySql（只是生产上不支持，但实际上支持）

|  |  |
| --- | --- |
| 优先级 | 优先级1 |

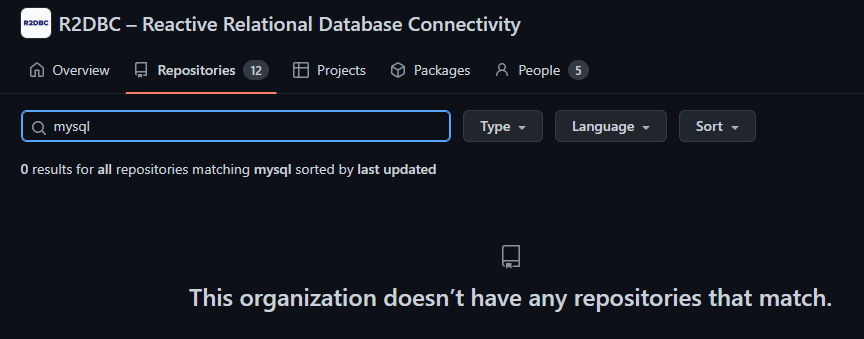
你可能会说可以用Spring官网的R2DBC，但请注意，R2DBC虽然支持mysql，但该项目已经很久没有维护了





况且，你需要注意，该项目是由个人开发，本身就存在风险





* 1. 简单看看 WebFlux 是如何分发请求的
     1. Spring MVC 的前端控制器是 DispatcherServlet, 而 WebFlux 是 DispatcherHandler，它实现了 WebHandler 接口
        + WebHandler 接口

public interface WebHandler **{**Mono**<**Void**>**handle**(**ServerWebExchange **exchange);  
}**

* + - * 查看DispatcherHandler类中处理请求的 handle 方法

@Override  
public Mono**<**Void**>**handle**(**ServerWebExchange **exchange) {**if **(**this**.**handlerMappings **==** null**) {**return createNotFoundError**();  
  }**if **(**CorsUtils**.***isPreFlightRequest***(exchange.**getRequest**())) {**return handlePreFlight**(exchange);  
  }**return Flux**.***fromIterable***(**this**.**handlerMappings**)  
      .**concatMap**(mapping** -> **mapping.**getHandler**(**exchange**))  
      .**next**()  
      .**switchIfEmpty**(**createNotFoundError**())  
      .**flatMap**(handler** -> invokeHandler**(**exchange**, handler))  
      .**flatMap**(result** -> handleResult**(**exchange**, result));  
}**

* + - * + ①：ServerWebExchange 对象中放置每一次 HTTP 请求响应信息，包括参数等；
        + ②： 判断整个接口映射 mappings 集合是否为空，空则创建一个 Not Found 的错误；
        + ③： 根据具体的请求地址获取对应的 handlerMapping;
        + ④： 调用具体业务方法，也就是我们定义的接口方法；
        + ⑤： 处理返回的结果；
  1. 快速入门
     1. 全局Router API方式
        + 添加依赖

<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-webflux</artifactId>  
</dependency>  
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-test</artifactId>  
    <scope>test</scope>  
</dependency>  
<dependency>  
    <groupId>io.projectreactor</groupId>  
    <artifactId>reactor-test</artifactId>  
    <scope>test</scope>  
</dependency>

* + - * 启动类

@SpringBootApplication  
public class WebfluxApplication **{**public static void main**(**String**[] args)** throws Exception **{**SpringApplication**.***run***(**WebfluxApplication**.**class**, args);  
  }  
  
}**

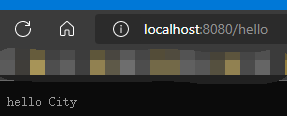
* + - * 添加Handler

@Component  
public class CityHandler **{**public Mono**<**ServerResponse**>**helloCity**(**ServerRequest **request) {**return ServerResponse**.***ok***().**contentType**(**MediaType**.***TEXT\_PLAIN***).**body**(**BodyInserters**.***fromValue***(**"hello City"**));  
    }  
  
}**

* + - * 添加Router

@Configuration  
public class CityRouter **{**@Bean  
    public RouterFunction**<**ServerResponse**>**routeCity**(**CityHandler **cityHandler) {**return RouterFunctions**.***route***(**RequestPredicates**.***GET***(**"/hello"**).**and**(**RequestPredicates**.***accept***(**MediaType**.***TEXT\_PLAIN***)), cityHandler**::helloCity**);  
    }  
  
}**

* + - * 启动启动类，访问http://localhost:8080/hello



* + 1. 使用SpringMVC那套注解
       - 添加依赖

<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-webflux</artifactId>  
</dependency>  
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-test</artifactId>  
    <scope>test</scope>  
</dependency>  
<dependency>  
    <groupId>io.projectreactor</groupId>  
    <artifactId>reactor-test</artifactId>  
    <scope>test</scope>  
</dependency>

* + - * 添加启动类

@SpringBootApplication  
public class WebfluxApplication **{**public static void main**(**String**[] args)** throws Exception **{**SpringApplication**.***run***(**WebfluxApplication**.**class**, args);  
  }  
  
}**

* + - * 添加Controller

@RestController  
public class CityController **{**@GetMapping**(**"hello"**)**public String hello**() {**return "Hello, WebFlux!!!"**;  
    }**@GetMapping**(**"user"**)**public Mono**<**City**>**getCity**() {**City city **=** new **City();**city**.**setId**(**12345678L**);**city**.**setName**(**"beijing"**);**return Mono**.***just***(**city**);  
    }  
  
}**

* + - * 启动项目，访问http://localhost:8080/user
      * Webflux的server-sent event

@GetMapping**(**value **=** "flux"**,** produces **=** MediaType**.***TEXT\_EVENT\_STREAM\_VALUE***)**public Flux**<**String**>**flux**() {**return Flux**.***fromStream***(**IntStream**.***range***(**1**,** 5**).**mapToObj**(value** -> **{**try **{**TimeUnit**.***SECONDS***.**sleep**(**1**);  
        }** catch **(**InterruptedException **e) {**throw new **RuntimeException(e);  
        }**return "flux data--" **+ value;  
    }));  
}**

* + - * + 这样做会告诉客户端，我服务端的请求是一个流，请客户端不要关闭请求链接，等待下一个包
      * 编写测试类

@RunWith**(**SpringRunner**.**class**)**@WebFluxTest**(**controllers **=** CityController**.**class**)**public class DemoApplicationTests **{**@Resource  
    private WebTestClient webTestClient**;**@Test  
    public void getAllMessagesShouldBeOk**()** throws Exception **{**webTestClient**.**get**().**uri**(**"/user"**)  
                .**exchange**().**expectStatus**().**isOk**();  
    }  
  
}**

* + - * + 需要额外导入

<dependency>  
    <groupId>junit</groupId>  
    <artifactId>junit</artifactId>  
    <scope>test</scope>  
</dependency>

1. WebFlux Web CRUD 实践
   1. mysql crud
      1. 思路
         * Spring WebFlux 使用 R2DBC 访问 MySQL
           + https://github.com/mirromutth/r2dbc-mysql这个项目很久没有更新了，找了个新的non-blocking驱动
           + https://github.com/jasync-sql/jasync-sql
      2. 步骤
         * 导入依赖

<dependencies>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-webflux</artifactId>  
    </dependency>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-data-r2dbc</artifactId>  
    </dependency>  
    <dependency>  
        <groupId>com.github.jasync-sql</groupId>  
        <artifactId>jasync-r2dbc-mysql</artifactId>  
        <version>2.1.1</version>  
    </dependency>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-test</artifactId>  
        <scope>test</scope>  
    </dependency>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-actuator</artifactId>  
    </dependency>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-devtools</artifactId>  
        <scope>provided</scope>  
        <optional>true</optional>  
    </dependency>  
    <dependency>  
        <groupId>io.projectreactor</groupId>  
        <artifactId>reactor-test</artifactId>  
        <scope>test</scope>  
    </dependency>  
    <dependency>  
        <groupId>org.projectlombok</groupId>  
        <artifactId>lombok</artifactId>  
        <scope>provided</scope>  
        <optional>true</optional>  
    </dependency>  
    <dependency>  
        <groupId>junit</groupId>  
        <artifactId>junit</artifactId>  
        <scope>test</scope>  
    </dependency>  
</dependencies>

* + - * application.yml

server**:**port**:** 8080  
spring**:**application**:**name**:** webflux-springmvc  
  r2dbc**:**username**:** root  
    password**:** 123456  
    url**:** r2dbc:mysql://127.0.0.1:3306/test?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUnicode=true&useSSL=false&serverTimezone=UTC

* + - * 启动类

@SpringBootApplication  
public class WebfluxMvcApplication **{**public static void main**(**String**[] args)** throws Exception **{**SpringApplication**.***run***(**WebfluxMvcApplication**.**class**, args);  
  }  
  
}**

* + - * 配置类

@Configuration  
@EnableTransactionManagement  
public class MysqlConfig **{**@Bean  
    public ConnectionFactory connectionFactory**(**R2dbcProperties **properties) {**ConnectionFactoryOptions options **=** ConnectionFactoryOptions**.***parse***(properties.**getUrl**());**com**.**github**.**jasync**.**sql**.**db**.**Configuration configuration **=** new com**.**github**.**jasync**.**sql**.**db**.Configuration(  
                properties.**getUsername**(),**Objects**.***requireNonNull***(**options**.**getValue**(**ConnectionFactoryOptions**.***HOST***)).**toString**(),**Integer**.***parseInt***(**String**.***valueOf***(**options**.**getValue**(**ConnectionFactoryOptions**.***PORT***))),  
                properties.**getPassword**(),**Objects**.***requireNonNull***(**options**.**getValue**(**ConnectionFactoryOptions**.***DATABASE***)).**toString**());**// 创建 JasyncConnectionFactory 对象  
        return new **JasyncConnectionFactory(**new **MySQLConnectionFactory(**configuration**));  
    }**@Bean  
    public ReactiveTransactionManager transactionManager**(**R2dbcProperties **properties)** throws URISyntaxException **{**return new **R2dbcTransactionManager(**this**.**connectionFactory**(properties));  
    }  
  
}**

* + - * + 没有这个配置类就会报错

ClassCastException: class java.lang.String cannot be cast to class java.time.Duration (java.lang.String and java.time.Duration are in module java.base of loader 'bootstrap')

graphic

这是个 String 类型在 kotlin中使用 as 转化成 Duration 会报错正常

应该使用 as? Duration?

这段配置类是需要的，按照原先的配置，那么就会报错

报错原因是类型不兼容，不知道怎么解决，可能是包不兼容，说句实在话，我也不知道它为什么这么写源码

github上相关问题：

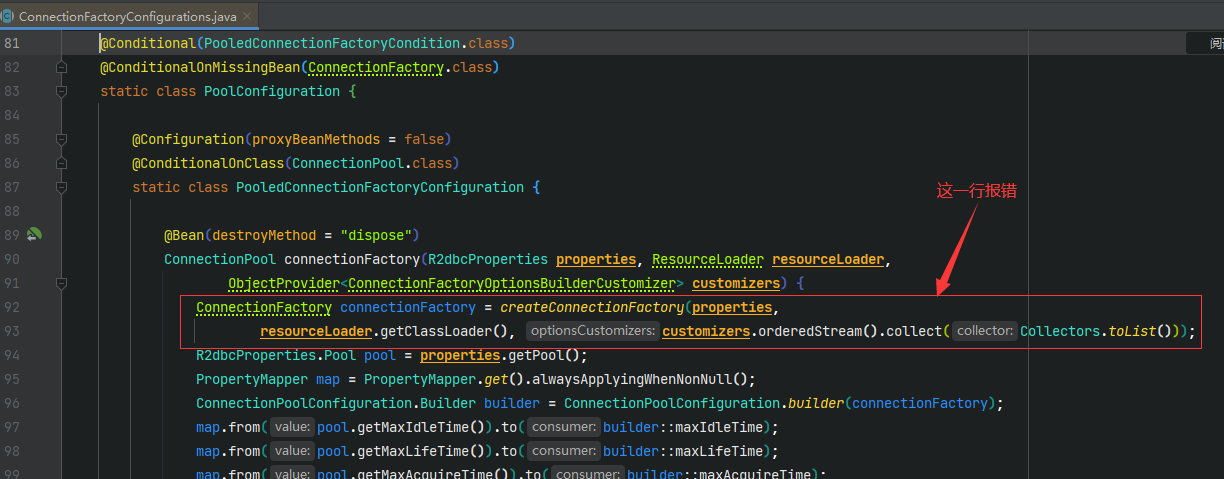
https://github.com/jasync-sql/jasync-sql/pull/307

https://github.com/jasync-sql/jasync-sql/issues/243

虽然没在github上找到如何解决的方法，但是我们可以重写

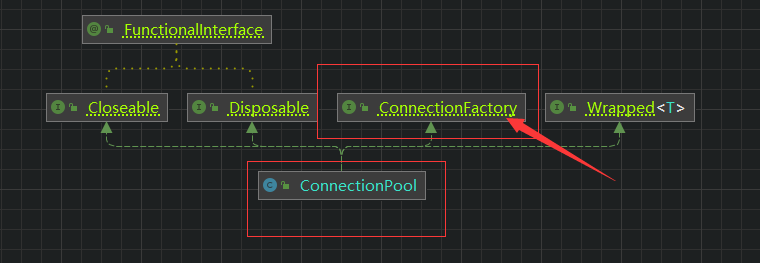
跟着错误栈，找到这个类

* + - * + 我们可以根据错误的调用栈找到这里

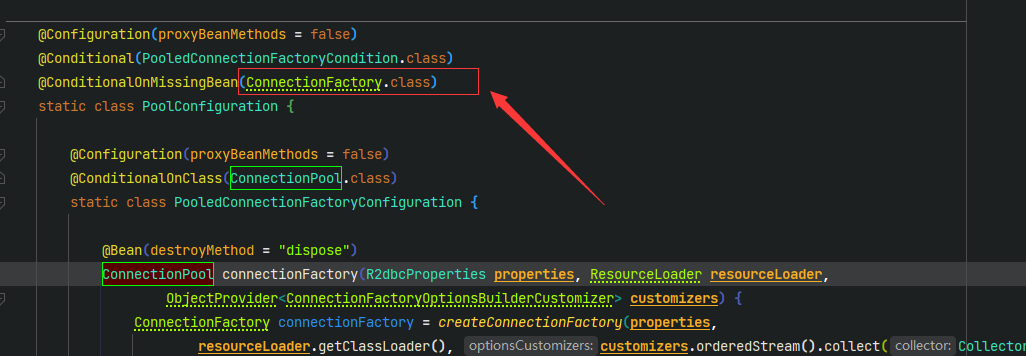


根据前面我们学过的@Bean必须要特化，所以这里是ConnectionPool

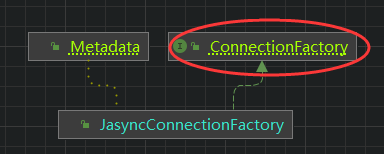
但实际上它是



从这里我们也能判断的出来



* + - * + 根据我们找到的Bean配置，知道需要配置一个属于自己的ConnectionFactory或者ConnectionPool



* + - * domain

@Data  
@Table**(**"users"**)**public class Users implements Serializable **{**@Id  
    private Long id**;**private Integer age**;**private String name**;  
}**

* + - * repository

@Repository  
public interface UsersRepository extends ReactiveCrudRepository**<**Users**,** Long**> {  
}**

* + - * controller

@RestController  
@Slf4j  
public class UsersController **{**@Resource  
    private UsersRepository usersRepository**;**@GetMapping**(**"get/{id}"**)**public Mono**<**Users**>**get**(**@PathVariable**(**"id"**)** Long **id) {**return usersRepository**.**findById**(id);  
    }  
  
  
}**

* + 1. 这样我们就能完成Spring Webflux访问mysql的功能了