聊聊 Spring Boot 2.x 那些事儿

本文目录：

大家看到目录，这么多内容，简直一本书的节奏。如果很详细，确实是。 可是只有一篇文章我大致讲讲每个点，其是什么，其主要业界的使用场景是什么，然后具体会有对应的博客教程。恩，下面我们聊聊 Spring 2.0。

一、即将的 Spring 2.0

spring.io 官网有句醒目的话是：

BUILD ANYTHING WITH SPRING BOOT

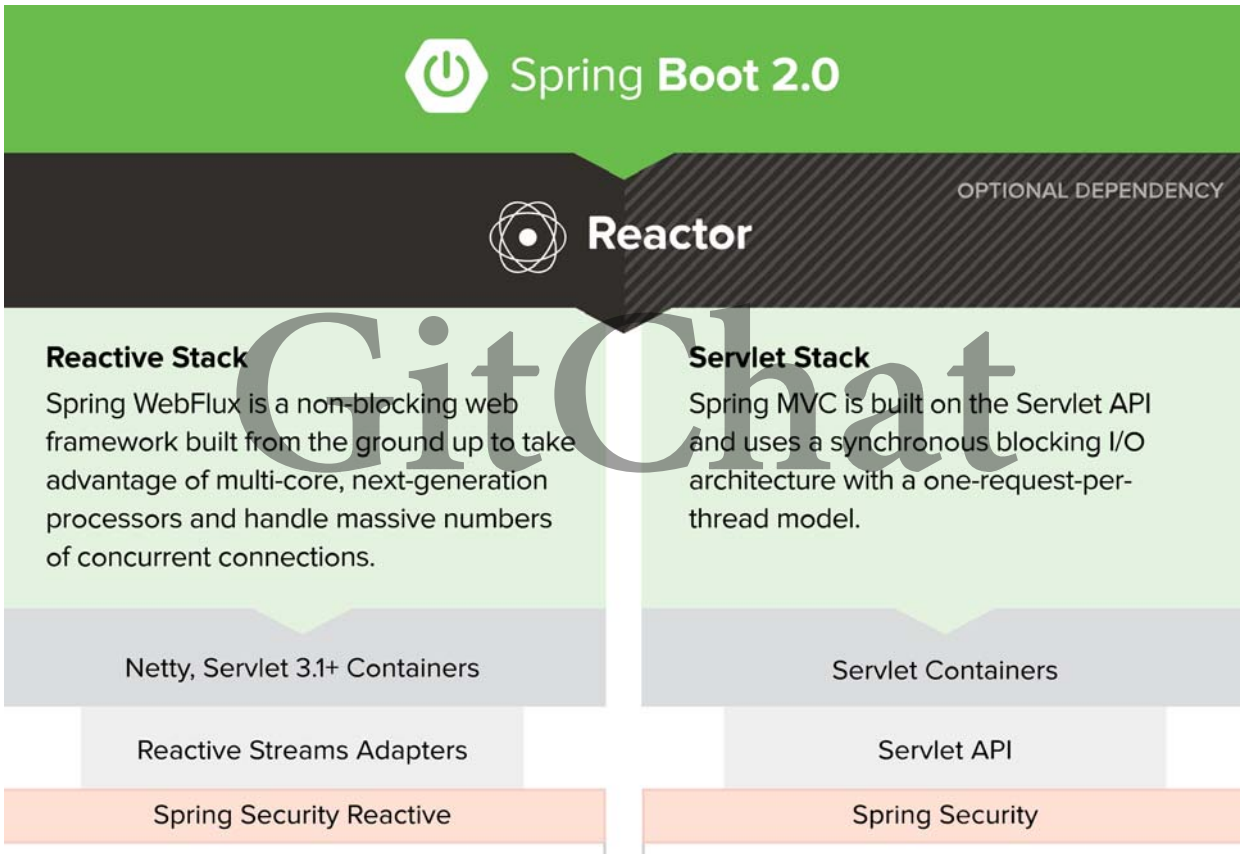
Java 程序员都知道 Spring 是什么，Spring 走过了这么多个年头。Spring 是 Java 应用程序平台开发框架，肯定也是跨平台的。同样，它也是 Java EE 轻量级框架，为 Java 开发这

```
Spring Boot makes it easy to create stand-alone, production-grade
Spring based Applications that you can “just run”... Most Spring
Boot applications need very little Spring configuration.
```

Spring Boot(英文中是“引导”的意思)，是用来简化Spring应用的搭建到开发的过程。应用开箱即用，只要通过 “just run”（可能是 java -jar 或 tomcat 或 maven插件run 或 shell脚本），就可以启动项目。二者，Spring Boot 只要很少的Spring配置文件（例如那些xml，property）。

因为“习惯优先于配置”的原则，使得Spring Boot在快速开发应用和微服务架构实践中得到广泛应用。

Spring 目前是 2.0.0 M5 版本，马上 2.0 release 了。如图：Spring 2.0 架构图



```
响应式 Stream
http://www.reactive-streams.org/
webflux 官方文档
https://docs.spring.io/spring/docs/current/spring-framework-
reference/web-reactive.html#spring-webflux
```

2. 开发环境和 IDE

大致介绍了 Spring Boot 2.0 是什么，下面我们快速入门下 Spring Boot 2.0。常言道，磨刀不误砍柴工砍柴工。在搭建一个 Spring Boot 工程应用前，需要配置好开发环境及安装好开发工具：

- JDK 1.8+
  Spring Boot 2.x 要求 JDK 1.8 环境及以上版本。另外，Spring Boot 2.x 只兼容 Spring Framework 5.0 及以上版本。

- Maven 3.2+
  为 Spring Boot 2.x 提供了相关依赖构建工具是 Maven，版本需要 3.2 及以上版本。使用 Gradle 则需要 1.12 及以上版本。Maven 和 Gradle 大家各自挑选下喜欢的就好。

- IntelliJ IDEA
  IntelliJ IDEA（简称 IDEA）是常用的开发工具，也是本书推荐使用的。同样使用 Eclipse IDE 自然也是可以的。

这里额外介绍下，对于 Java 新手或者刚刚认识 Spring 的小伙伴。Spring Boot CLI 是一个学习 Spring Boot 的很好的工具。Spring Boot CLI 是 Spring Boot Commad Line 的缩写，是 Spring Boot 命令行工具。在 Spring Boot CLI 可以跑 Groovy 脚本，通过简单的 Java 语法就可以快速而又简单的学习 Spring Boot 原型。

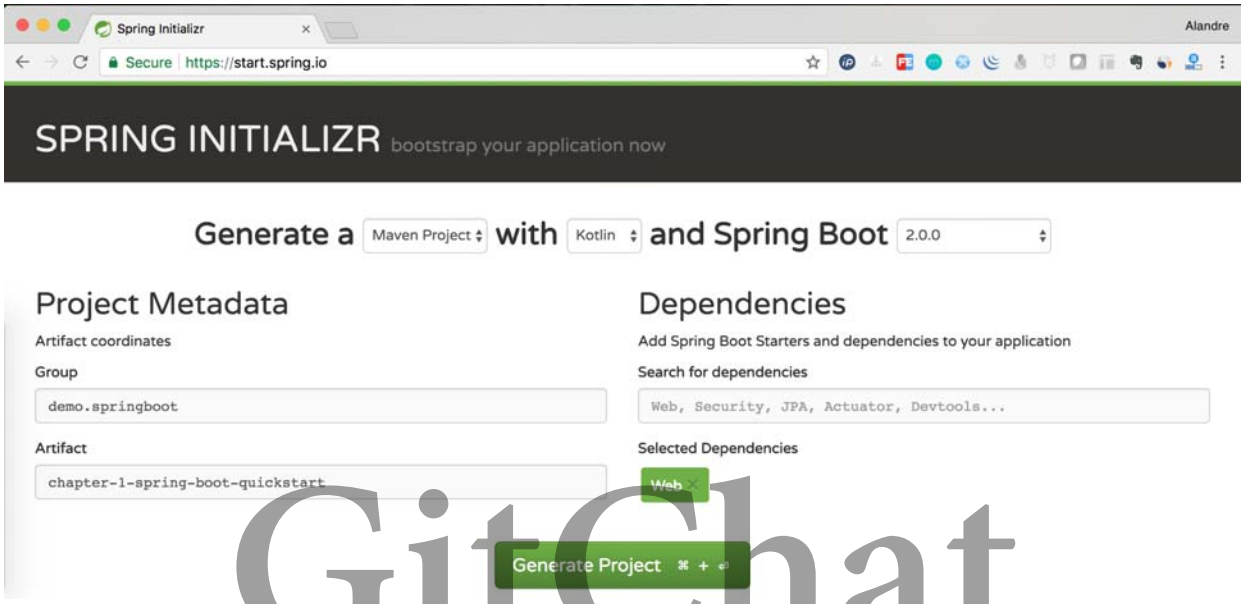
Spring Boot CLI 具体快速入门看下我写的地址：https://www.bysocket.com/?p=1982

那最常用、最推荐的入门当然是使用 Spring Initializr，下面介绍下如何使用 Spring

第二步，输入 Maven 工程信息，即项目组 `groupId` 和名字 `artifactId` 。这里对应 Maven 信息为：

- groupId：demo.springboot
- artifactId：spring-boot-quickstart
  这里默认版本号 version 为 0.0.1-SNAPSHOT 。三个属性在 Maven 依赖仓库是唯一标识的。

第三步，选择工程需要的 Starter 组件和其他依赖。最后点击生成按钮，即可获得骨架工程压缩包。如图 ：



在对应的 *Application 增加对应的代码如下：（这来自是官方 demo ）

```
@Controller
@EnableAutoConfiguration
public class Application {

    @RequestMapping("/")
    @ResponseBody
    String home() {
```

```
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on
port(s): 8080 (http)
2017-10-15 10:05:20.000  INFO 17963 --- [           main]
demo.springboot.QuickStartApplication    : Started
QuickStartApplication in 5.544 seconds (JVM running for 6.802)
```

访问地址 localhost:8080 ，成功返回 “Hello World!” 的字符串。

这就是 Spring Boot 使用，是不是很方便。介绍下工程的目录结构：

```
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── demo
    │   │       └── springboot
    │   │           └── Application.java
    │   └── resources
    │       ├── application.properties
    │       ├── static
    │       └── templates
    └── test
        └── java
            └── demo
                └── springboot
                    └── QuickstartApplication Tests.java
```


这是默认的工程结构，java 目录中是编写代码的源目录，比如三层模型大致会新建三个包目录，web 包负责 web 服务，service 包负责业务逻辑，domain 包数据源服务。对应 java 目录的是 test 目录，编写单元测试的目录。

resources 目录会有 application.properties 应用配置文件，还有默认生成的 static 和 templates 目录，static 用于存放静态资源文件，templates 用于存放模板文件。可以在 application.properties 中自定义配置资源和模板目录

开发中，很多功能是通过添加 Starter 组件的方式来进行实现。下面都是常用的组件，还有很多事务、消息、安全、监控、大数据等支持，这里就不介绍了。大家可以同理可得去学习哈。

1. Web：REST API & 模板引擎

在 Web 开发中，常见的场景有传统的 Web MVC 架构和前后端分离架构。

Web MVC 架构

Web MVC 模式很适合 Spring Boot 来开发，View 使用 JSP 或者其他模板引擎（默认支持：FreeMarker 、 Groovy 、 Thymeleaf 、 Mustache ）。

传统模式比如获取用户，是从用户 view 层发送获取用户请求到 Spring Boot 提供的用户控制层，然后获取数据封装进 Model，最后将 model 返回到 View。因为 Spring Boot 基于 Spring ，所以 Spring 能做的，Spring Boot 就能做，而且更加方便，更近快速。

前后端分离架构

前后端分离架构，免不了的是 API 文档作为中间的桥梁。Spring Boot 很方便的开发 REST API，前端通过调用 REST API 获取数据。数据形式可能是 JSON 或者 XML 等。然后进行视图渲染，这里前端也有对应的前端模板引擎。其实 H5 ，PC，APP 都可采取类似的方式实现。这里的 API 文档可以使用 Swagger2 或者 APIDOC 来实现。

那具体聊聊 REST API

RESTful是什么？RESTful（Representational State Transfer）架构风格，是一个Web自身的架构风格，底层主要基于HTTP协议（ps:提出者就是HTTP协议的作者），是分布式应用架构的伟大实践理论。RESTful架构是无状态的，表现为请求-响应的形式，有别于基于Bower的SessionId不同。Spring Boot 的注解 `@RestController` 支持实现 RESTful 控制层。

那有个问题？权限怎么控制？
RESTful是无状态的，所以每次请求就需要对起进行认证和授权。

有人在我博客上评论 模板语言 现在没人用了吧。我们还是先了解下，什么是模板语言再说吧。常见的模板语言都包含以下几个概念：数据（Data）、模板（Template）、模板引擎（Template Engine）和结果文档（Result Documents）。

- 数据
  数据是信息的表现形式和载体，可以是符号、文字、数字、语音、图像、视频等。数据和信息是不可分离的，数据是信息的表达，信息是数据的内涵。数据本身没有意义，数据只有对实体行为产生影响时才成为信息。

- 模板
  模板，是一个蓝图，即一个与类型无关的类。编译器在使用模板时，会根据模板实参对模板进行实例化，得到一个与类型相关的类。

- 模板引擎
  模板引擎（这里特指用于Web开发的模板引擎）是为了使用户界面与业务数据（内容）分离而产生的，它可以生成特定格式的文档，用于网站的模板引擎就会生成一个标准的HTML文档。

- 结果文档
  一种特定格式的文档，比如用于网站的模板引擎就会生成一个标准的HTML文档。

模板语言用途广泛，常见的用途如下：

- 页面渲染
- 文档生成
- 代码生成
- 所有 “数据+模板=文本” 的应用场景

所以大家看到这个用途，应该不会说没有用了吧。具体按模板语言 Thymeleaf 为例，使用如下

pom.xml Thymeleaf 依赖

使用模板引擎，就在 pom.xml 加入 Thymeleaf 组件依赖：

```
spring.thymeleaf.cache=true # Enable template caching.
spring.thymeleaf.check-template=true # Check that the template
exists before rendering it.
spring.thymeleaf.check-template-location=true # Check that the
templates location exists.
spring.thymeleaf.enabled=true # Enable Thymeleaf view resolution
for Web frameworks.
spring.thymeleaf.encoding=UTF-8 # Template files encoding.
spring.thymeleaf.excluded-view-names= # Comma-separated list of
view names that should be excluded from resolution.
spring.thymeleaf.mode=HTML5 # Template mode to be applied to
templates. See also StandardTemplateModeHandlers.
spring.thymeleaf.prefix=classpath:/templates/ # Prefix that gets
prepended to view names when building a URL.
spring.thymeleaf.reactive.max-chunk-size= # Maximum size of data
buffers used for writing to the response, in bytes.
spring.thymeleaf.reactive.media-types= # Media types supported by
the view technology.
spring.thymeleaf.servlet.content-type=text/html # Content-Type
value written to HTTP responses.
spring.thymeleaf.suffix=.html # Suffix that gets appended to view
names when building a URL.
spring.thymeleaf.template-resolver-order= # Order of the template
resolver in the chain.
spring.thymeleaf.view-names= # Comma-separated list of view names
that can be resolved.
```


Controller 的使用方式，和以前的 Spring 方式一致，具体的Tymeleaf 的语法糖，大家可以看看官方文档http://www.thymeleaf.org/documentation.html。本案例具体整合教程：https://www.bysocket.com/?p=1973。

2. Data：JPA 、 Mybatis

Data，顾名思义是数据。数据存储有 SQL 和 NoSQL：

Spring Boot Mybatis 依赖 mybatis-spring-boot-starter

Mybatis 也是业界互联网流行的数据操作层框架。有两种形式进行使用 Mybatis。第一、纯 Annotation，第二、使用 xml 配置 SQL。个人推荐使用 xml 配置 SQL， SQL 和业务代码应该隔离，方便和 DBA 校对 SQL。二者 XML 对较长的 SQL 比较清晰。

虽然 XML 形式是我比较推荐的，但是注解形式也是方便的。尤其一些小系统，快速的 CRUD 轻量级的系统。这里可见，mybatis-spring-boot-starter依赖是非官方提供的。

Springboot 整合 Mybatis 的完整 Web 案例教程：https://www.bysocket.com/?p=1610

Spring Boot 整合 Mybatis Annotation 注解的完整 Web 案例教程：https://www.bysocket.com/?p=1811

另外，搜索常用 ES，spring-data-elasticsearch 是 Spring Data 的 Community modules 之一，是 Spring Data 对 Elasticsearch 引擎的实现。 Elasticsearch 默认提供轻量级的 HTTP Restful 接口形式的访问。相对来说，使用 HTTP Client 调用也很简单。

但 spring-data-elasticsearch 可以更快的支持构建在 Spring 应用上，比如在 application.properties 配置 ES 节点信息和 spring-boot-starter-data-elasticsearch 依赖，直接在 Spring Boot 应用上使用。

我也写了点系列博客在：https://www.bysocket.com/?tag=elasticsearch

还有很多组件无法一一介绍了，比如常用的 Redis 做缓存操作等。

三、生产指标监控 Actuator

Starter 组件 Actuator 提供了生产级监控的特性，可以使用 Actuator 来监控应用的一切。使用方式也很简单，加入对应的依赖，然后访问各个 HTTP 请求就可以获取需要的信息。具体提供的信息有：

这里我师弟写一遍教程不错：http://www.jianshu.com/p/7ecb57a3f326

四、内嵌式容器 Tomcat / Jetty / Undertow

Spring Boot 运行的应用是独立的一个 Jar 应用，实际上在运行时启动了应用内部的内嵌容器，容器初始化 Spring 环境及其组件并启动应用。但我们也可以自定义配置 内嵌式容器，比如将 Tomcat 换成 Jetty。 Spring Boot 能这样工作，主要靠下面两点：自动配置和外化配置。

自动配置

Spring Boot 在不需要任何配置情况下，就直接可以运行一个应用。实际上，Spring Boot 框架的 `spring-boot-autoconfigure` 依赖做了很多默认的配置项，即应用默认值。这种模式叫做 “自动配置”。Spring Boot 自动配置会根据添加的依赖，自动加载依赖相关的配置属性并启动依赖。例如，默认用的内嵌式容器是 Tomcat ，端口默认设置为 8080。

外化配置

Spring Boot 简化了配置，在 application.properties 文件配置常用的应用属性。Spring Boot 可以将配置外部化，这种模式叫做 “外化配置”。将配置从代码中分离外置，最明显的作用是只要简单地修改下外化配置文件，就可以在不同环境中，可以运行相同的应用代码。

所以，Spring Boot 启动应用，默认情况下是自动启动了内嵌容器 Tomcat，并且自动设置了默认端口为 8080。另外还提供了对 Jetty、Undertow 等容器的支持。开发者自行在添加对应的容器 Starter 组件依赖，即可配置并使用对应内嵌容器实例。具体操作如下：

第一步，将 tomcat 依赖排除：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

上面大致介绍了 Spring Boot 2.0 有的那些特性，也就是那些事，最后入门介绍下 WebFlux。

五、Spring 5 & Spring WebFlux

最后，Spring Boot 2.0 支持 Spring 5，Spring 5 具有了强大的特性：WebFlux/Reactor。所以最后来聊聊 WebFlux ，就算入个门吧。

对照下 Spring Web MVC ，Spring Web MVC 是基于 Servlet API 和 Servlet 容器设计的。那么 Spring WebFlux 肯定不是基于前面两者，它基于 Reactive Streams API 和 Servlet 3.1+ 容器设计。

那 Reactive Streams API 是什么？

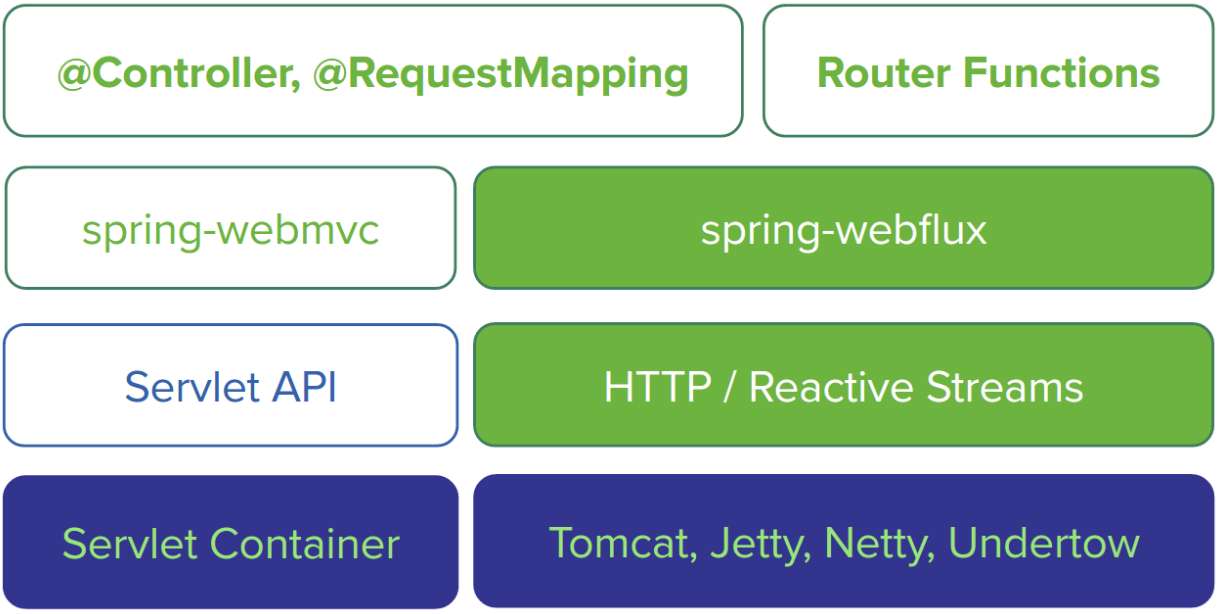
先理解 Stream 流是什么？流是序列，是生产者生产，一个或多个消费者消费的元素序列。这种具体的设计模式成为发布订阅模式。常见的流处理机制是 pull / push 模式。背压是一种常用策略，使得发布者拥有无限制的缓冲区存储 item，用于确保发布者发布 item 太快时，不会去压制订阅者。

Reactive Streams （响应式流）是提供处理非阻塞背压异步流的一种标准。主要针对的场景是运行时环境（包括 JVM 和 JS）和网络。同样，JDK 9 java.util.concurrent 包提供了两个主要的 API 来处理响应流：

- Flow
- SubmissionPublisher

为啥只能运行在 Servlet 3.1+ 容器？

Spring WebFlux 是什么



先看这张图，上面我们了解了容器、响应流。这里介绍下 Spring WebFlux 是什么？Spring WebFlux 是 Spring 5 的一个新模块，包含了响应式 HTTP 和 WebSocket 的支持，另外在上层服务端支持两种不同的编程模型：

- 基于 Spring MVC 注解 @Controller 等
- 基于 Functional 函数式路由

下面是两个实现小案例，首先在 pom.xml 加入对应的依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

基于 Spring MVC 注解 RESTful API

```
        @PostMapping("/person")
        Mono<Void> create(@RequestBody Publisher<Person>
personStream) {
                return this.repository.save(personStream).then();
        }

        @GetMapping("/person")
        Flux<Person> list() {
                return this.repository.findAll();
        }

        @GetMapping("/person/{id}")
        Mono<Person> findById(@PathVariable String id) {
                return this.repository.findOne(id);
        }
}
```

但是 PersonRepository 这种 Spring Data Reactive Repositories 不支持 MySQL，进一步也不支持 MySQL 事务。所以用了 Reactivey 原来的 spring 事务管理就不好用了。jdbc jpa 的事务是基于阻塞 IO 模型的，如果 Spring Data Reactive 没有升级 IO 模型去支持 JDBC，生产上的应用只能使用不强依赖事务的。也可以使用透明的事务管理，即每次操作的时候以回调形式去传递数据库连接 connection。

Spring Data Reactive Repositories 目前支持 Mongo、Cassandra、Redis、Couchbase 。

如果应用只能使用不强依赖数据事务，依旧使用 MySQL ，可以使用下面的实现，代码如下：

```
@RestController
@RequestMapping(value = "/city")
public class CityRestController {

    @Autowired
    private CityService cityService;
```

```
    }

    @RequestMapping(method = RequestMethod.POST)
    public Mono<Long> createCity(@RequestBody City city) {
        return Mono.create(cityMonoSink ->
cityMonoSink.success(cityService.saveCity(city)));
    }

    @RequestMapping(method = RequestMethod.PUT)
    public Mono<Long> modifyCity(@RequestBody City city) {
        return Mono.create(cityMonoSink ->
cityMonoSink.success(cityService.updateCity(city)));
    }

    @RequestMapping(value = "/{id}", method =
RequestMethod.DELETE)
    public Mono<Long> modifyCity(@PathVariable("id") Long id) {
        return Mono.create(cityMonoSink ->
cityMonoSink.success(cityService.deleteCity(id)));
    }
}
```

findAllCity 方法中，利用 Flux.create 方法对响应进行创建封装成 Flux 数据。并且使用 lambda 写数据流的处理函数会十分的方便。

Service 层依旧是以前那套逻辑，业务服务层接口如下：

```
public interface CityService {

    /**
     * 获取城市信息列表
     *
     * @return
     */
    List<City> findAllCity();
```

```
    Long saveCity(City city);

    /**
     * 更新城市信息
     *
     * @param city
     * @return
     */
    Long updateCity(City city);

    /**
     * 根据城市 ID,删除城市信息
     *
     * @param id
     * @return
     */
    Long deleteCity(Long id);
}
```

具体案例在我的 Github：https://github.com/JeffLi1993/springboot-learning-example

基于 Functional 函数式路由实现 RESTful API

创建一个 Route 类来定义 RESTful HTTP 路由：

```
import static
org.springframework.web.reactive.function.server.RequestPredicate
s.GET;
import static
org.springframework.web.reactive.function.server.RequestPredicate
s.accept;
import static
org.springframework.web.reactive.function.server.RouterFunctions.
```

```
    cityService:: findAllCity).and(route(

    GET("/api/user/{id}").and(accept(MediaType.APPLICATION_JSON)),
    cityService:: findCityById)
                    );
        }
    }
```

RoouterFunction 类似 Spring Web MVC 的 @RequestMapping ，用来定义路由信息，每个路由会映射到一个处理方法，当接受 HTTP 请求时候会调用该处理方法。

创建 HttpServerConfig 自定义 Http Server，这里创建一个 Netty HTTP 服务器：

```
    import org.springframework.http.server.reactive.HttpHandler;
    import
    org.springframework.http.server.reactive.ReactorHttpHandlerAdapte
    r;
    import reactor.ipc.netty.http.server.HttpServer;

    @Configuration
    public class HttpServerConfig {
        @Autowired
        private Environment environment;

        @Bean
        public HttpServer httpServer(RouterFunction<?>
    routerFunction) {
            HttpHandler httpHandler =
    RouterFunctions.toHttpHandler(routerFunction);
            ReactorHttpHandlerAdapter adapter = new
    ReactorHttpHandlerAdapter(httpHandler);
            HttpServer server = HttpServer.create("localhost",
    Integer.valueOf(environment.getProperty("server.port")));
            server.newHandler(adapter);
            return server;
        }
```

础实践，精致地写了 demo 和讲解。最后还是谢谢。