# 邮件发送

邮件发送是一个非常常见的功能，如注册、登录验证，发送重要消息等都需要使用到邮件发送。

Sun公司提供了JavaMail用来实现邮件发送，但是配置繁琐；

Spring提供了JavaMailSender用来简化邮件配置，实现邮件发送；

Spring Boot则提供了MailSenderAutoConfiguration进一步简化了邮件配置，实现邮件发送。

## 术语

SMTP全称为Simple Mail Transfer Protocol，译作简单邮件传输协议，它定义了邮件客户端软件与SMTP服务器之间，以及SMTP服务器与SMTP服务器之间的通信规则。[如1875889780@qq.com](mailto:如1875889780@qq.com)用户通过邮件客户端发送邮件，先将邮件传输到腾讯的SMTP服务器，这个过程就使用了SMTP协议，然后腾讯的SMTP服务器将邮件投递到网易的SMTP服务器，这个过程依赖使用了SMTP协议；即SMTP用于接收邮件；

POP3全称为Post Office Protocol3，译作邮局协议，它定义了邮件客户端与POP3服务器之间的通信规则，当邮件到达网易的SMTP服务器之后，接收方需要登录邮件服务器查看邮件，这个使用就需要使用到POP3协议；邮件服务器商会为每一个用户提供专门的邮件存储空间，SMTP服务器收到邮件之后，将邮件保存在用户的邮件存储空间中，如果用户要读取邮件，就需要通过邮件服务商的POP3邮件服务器来完成；

IMAP协议是对POP3协议的扩展，功能更强，作用类似。

## 准备

以QQ邮箱为例，使用QQ邮箱发送邮件，首页你得有一个QQ邮箱账户。

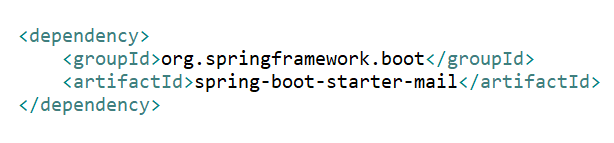
登陆QQ邮箱客户端，在设置中开启POP3/SMTP服务或者IMAP/SMTP服务。如下：



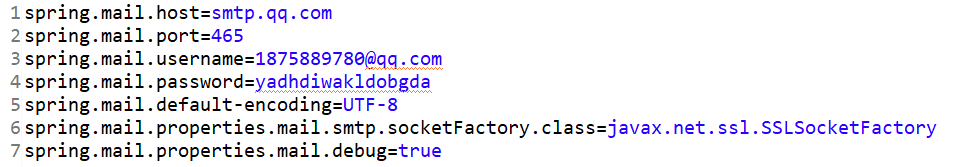
开启服务后，你会获得一个授权码，这个授权码在发送邮件时需要使用到。

## 配置

1. 添加spring-boot-starter-mail依赖，如下：



1. 配置application.properties中完成邮件基本信息配置，如下：



host配置邮箱服务器，

port配置端口号，因为使用的是qq邮箱，端口号可以是465或者587

username表示邮箱名；

password表示授权码；

default-encoding表示默认编码；

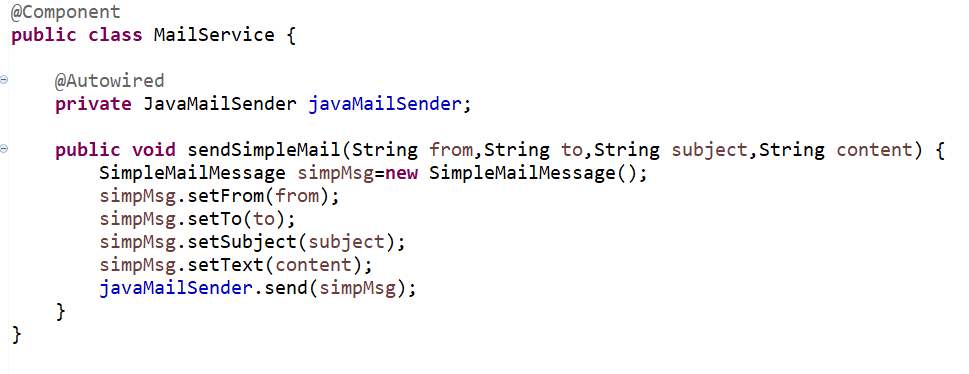
socketFactory.class表示SSL配置；

debug表示开启debug。

## 使用

### 发送简单邮件

1. 创建一个类来封装邮件的发送，如下：



JavaMailSender是Spring Boot提供的发送邮件所要使用的类；

方法形参from表示发送人、to表示收件人、subject表示邮件主题、content表示邮件内容。

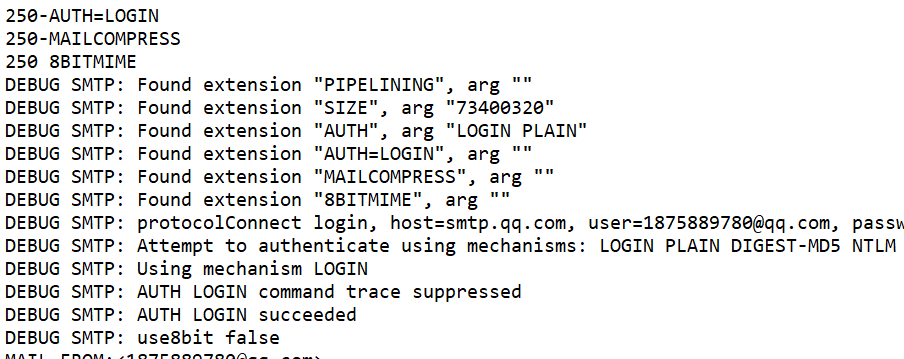
SimpleMailMessage实例表示一封邮件；

1. 编写测试，如下：



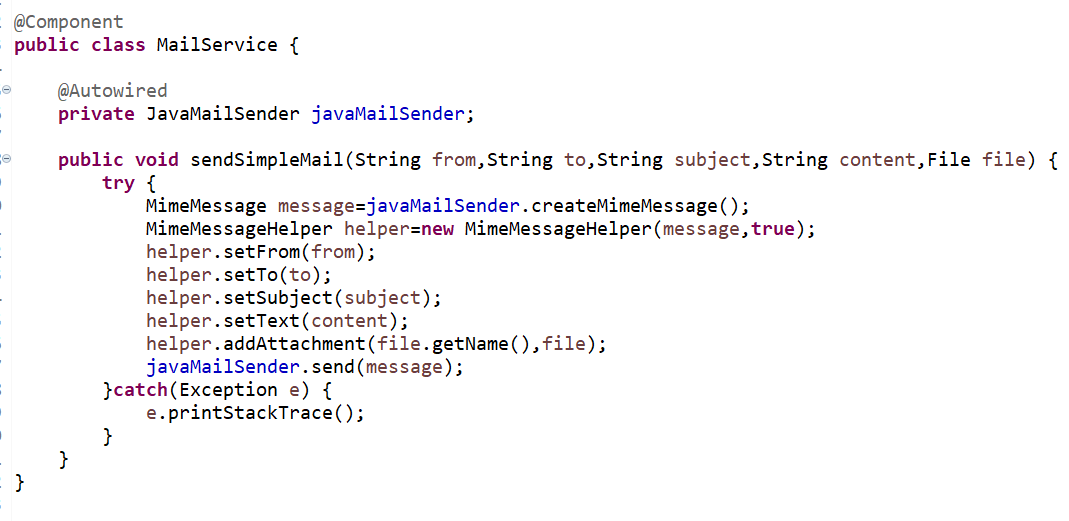
启动项目，访问路径，查看收件人邮箱是否收到邮件。

1. 由于配置了debug，所以在控制台也会打印排错信息，如下：



### 发送带附件的邮件

1. 发送一个带附件的邮件，只需要调用addAttachment方法即可，如果调用多次即可添加多个附件。如下：



这里使用了MimeMessageHelper简化了邮件配置，构造方法的第2个参数true表示使用multipart message类型的邮件，multipart message类型的邮件包含多个正文、附件以及内嵌资源，邮件的表现形式更加丰富；

通过调用addAttachement方法添加要发送的附件，该方法的第一个参数表示发送附件的文件名，第二个参数表示附件；

形参from表示发送人、to表示收件人、subject表示主题、content表示内容、file表示附件；

1. 进行测试，如下：



启动项目，访问路径，查看收件人邮箱是否收到邮件。

### 发送带图片资源的邮件

1）有的邮件正文中可能需要插入图片，使用FileSystemResource可以实现。如下：



形参srcPath表示图片资源路径，resIds表示图片资源Id；

setText方法的第二个参数true表示邮件正文是HTML格式的，如果不传默认为false，如果两者长度不相等，则退出，发送失败；

通过遍历取得图片资源路径以及调用addInline方法将资源加入到邮件中；

1. 进行测试，如下：

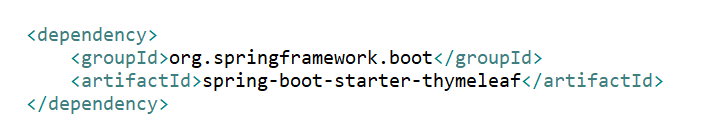


启动项目，登录收件人邮箱，查看邮件。

### 使用Thymeleaf构建邮件模板

通过调用setText可以发送HTML格式的正文，在HTML中可以嵌套我们网站的链接，以此可以达到用户注册后在邮箱中激活账户的目的，但拼凑HTML繁琐且不直观；为此我们可以将视图当做邮件发送给用户。在这里选择thymeleaf作为视图技术。

1. 添加thymeleaf依赖，如下：



1. 在resources/templates目录下创建邮件模板，如下：

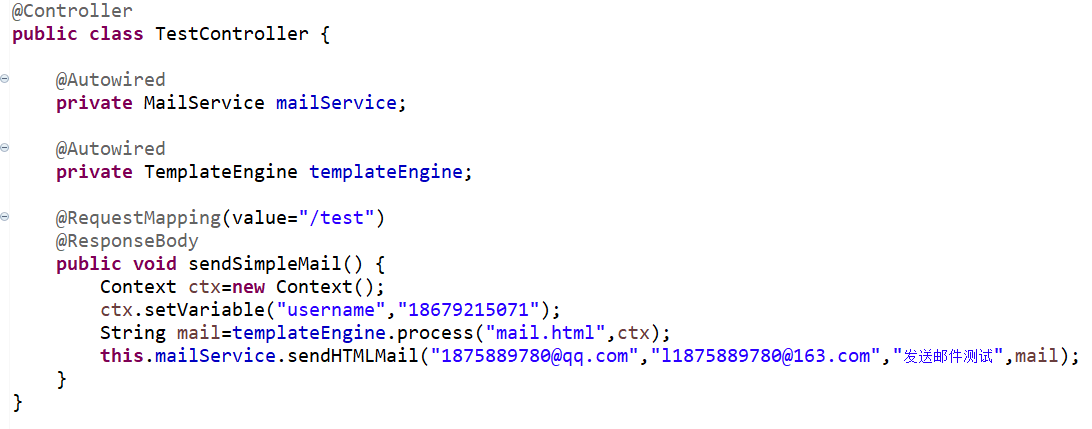


1. 发送邮件方法，如下：



由于发送的是视图，所以调用setText方法时必须通过true将邮件内容设置为html格式；

1. 编写测试，如下：



启动项目，访问路径，查看收件者邮箱是否收到邮件。

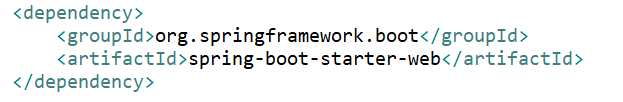
# 定时任务

定时任务是企业级开发常见的功能之一，如定时发送短信和邮件、定义数据备份、定义统计博客访客、定时统计订单数、定时删除某数据等；

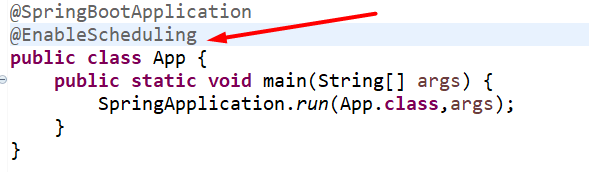
简单的定时任务可以直接通过Spring中的@Scheduled注解来实现，复杂的定时任务则可以通过集合Quartz来实现；

## @Scheduled

1. 添加spring-boot-startert-web依赖后即可使用@Scheduled注解，如下：



1. 在项目启动类上标注@EnableScheduling注解开启定时任务，如下：



1. 编写定时任务，如下：



@Scheduled(fixedDelay=10000) 表示该方法在项目启动后立即执行一次，之后每隔10秒执行一次；

@Scheduled(fixedRate=10000) 表示该方法在项目启动后不立即执行，而是每隔10秒执行一次；

@Scheduled(initialDelay=10000 , fixedRate=2000) 表示该方法在项目启动后，隔了10秒后执行一次，之后每隔2秒执行一次；

@Scheduled(cron=”0 \* \* \* \* ?”) 表示项目启动后，该方法每隔1分钟执行一次；cron指的是cron表达式；如果要了解cron表达式请查询相关资料；

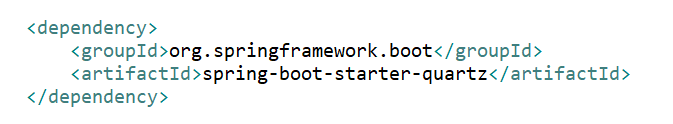
启动项目，观察控制台打印。

关于@Scheduled注解的更多用法可以查阅相关资料，关于定时任务也可以参考Java开发实战经典一书中关于定时任务的章节。

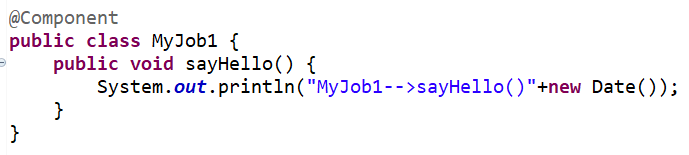
## Quartz

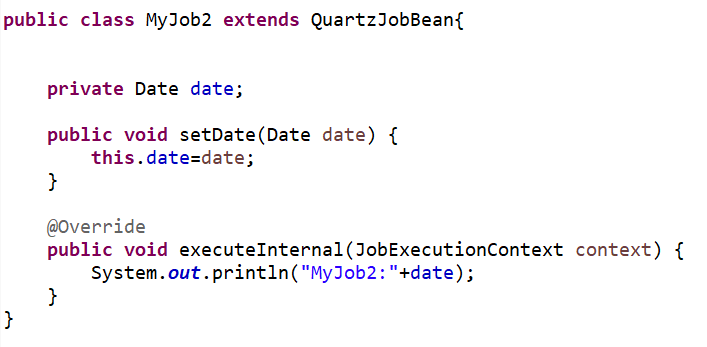
Quartz是一个丰富的开源作业调度库，由Java写成，可以集成在任何Java应用程序中。使用Quartz可以创建简单或者复杂的执行计划，它支持数据库、集群、插件以及邮件，并且也支持cron表达式；

1. 添加spring-boot-starter-quartz依赖，如下：



1. 创建两个Job，如下：

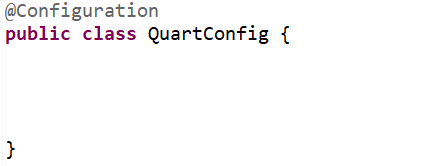




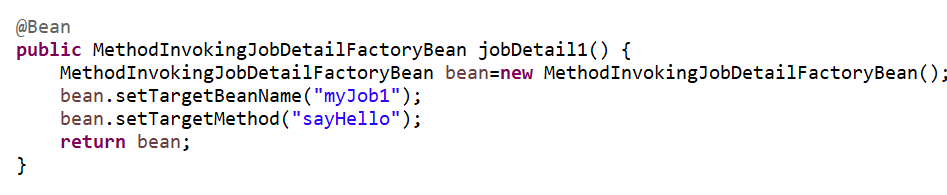
Job可以是一个普通的Bean，但需要使用@Component将其注册到Spring容器中；

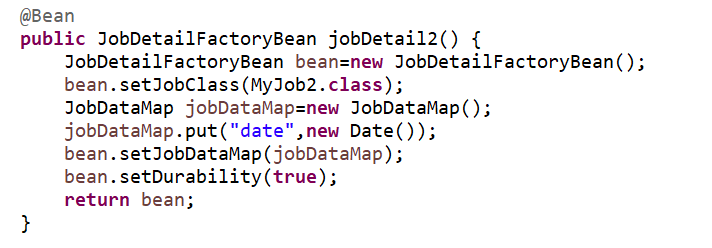
Job也可以通过继承QuartzJobBean抽象类来定义，但需要覆写该类中额executeInternal方法，该方法在任务调度时使用；

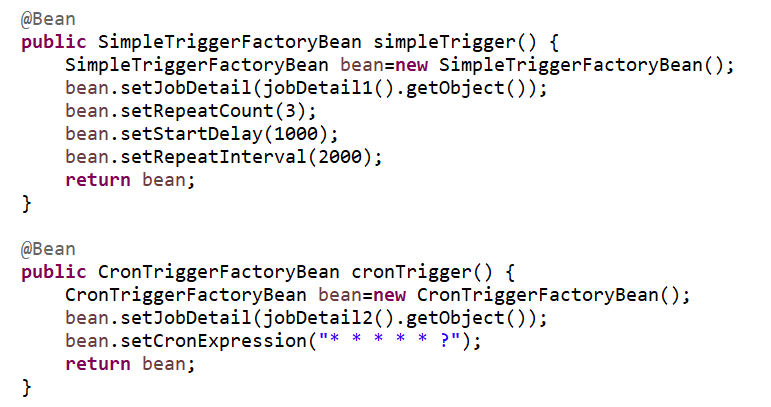
1. 创建Quartz配置类，对Job的Detail（细节）和Trigger（触发条件）进行配置，如下：

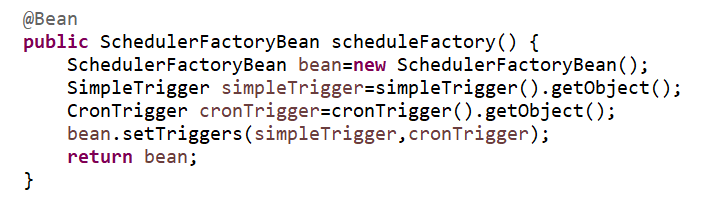


该配置类中定义了5个方法，如下：









配置JobDetail（Job细节）有两种方式：第一种是通过MethodInvokingJobDetailFactoryBean类配置，只需指定Job的实例名和调用的方法即可，但通过此种方式无法在配置JobDetail时传递参数；第二种方式是通过JobDetailFactoryBean来实现，这种方式只需要指定JobClass即可，通过这种方式可以在配置JobDetail时传递参数，Job中只需要提供属性名和该属性的setter方法。（讲解上述配置类中的jobDetail1方法和jobDetail2方法）

配置JobTrigger（Job触发条件）有多种不同的实现，在这里只展示了两种常用的Trigger：SimpleTrigger和CornTrigger；SimpleTrigger通过SimpleTriggerFactoryBean创建，在SimpleTriggerFactoryBean对象中，首先设置目标JobDetail，然后调用setRepeatCount方法配置任务的循环次数，调用setStartDelay方法配置任务启动延迟时间，调用setRepeatInterval配合任务的时间间隔；CronTrigger通过CronTriggerFactoryBean进行创建，在CronTriggerFactoryBean对象中，则主要配置JobDetail和Cron表达式；（讲解上述配置类中的simpleTrigger方法和cronTrigger方法）

最后通过SchedlerFactoryBean创建SchedulerFactory，然后配置Trigger即可。（讲解上述配置类中的scheduleFacotry方法）

1. 启动项目，观察打印台输出。

# 批处理

## 简介

Spring Batch是一个开源的、全面的、轻量级的批处理框架，通过Spring Batch可以实现强大的批处理应用程序的开发。

Spring Batch还提供记录/跟踪、事务管理、作用处理统计、作业重启以及资源管理等功能；

Spring Batch结合定时任务可以发挥更大的作用；

Spring Batch还提供了ItemReader、ItemProcessor和ItemWriter来完成数据的读取、处理以及写出操作，并且可以将批处理的执行状态持久化到数据库中。

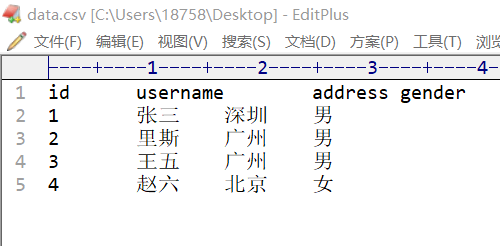
## 整合Spring Boot

在简介中已经讲解过Spring Batch还提供了ItemReader、ItemProcessor和ItemWriter来完成数据的读取、处理以及写出操作，并且可以将批处理的执行状态持久化到数据库中。

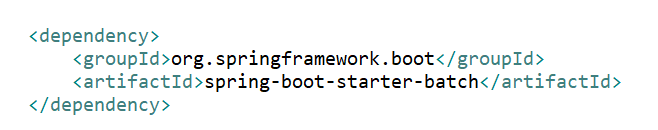
下面通过一个示例整合Spring Boot，将一个data.csv文件中的数据通过批处理读取，并保存到数据表中。

1. classpath路径下存在data.csv文件（关于csv文件请自行百度），文件内容如下：

（使用Unicode编码打开）



1. 添加spring-boot-start-batch依赖，如下：

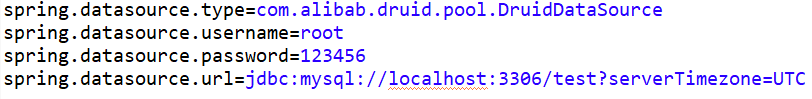


1. 因为我们需要将data.csv中的数据插入到数据库中，这里这里还需使用到持久层技术，添加相关依赖，如下：



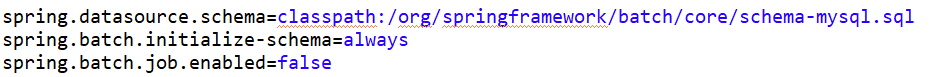
这里使用的持久层技术是Spring的JdbcTemplate；连接池使用的阿里巴巴的连接池；

1. 在application.properties中配置数据连接信息，如下：



配置了数据源、用户名、密码、以及使用的MySQL数据库地址；

1. 在application.properties中配置Spring Batch，如下：

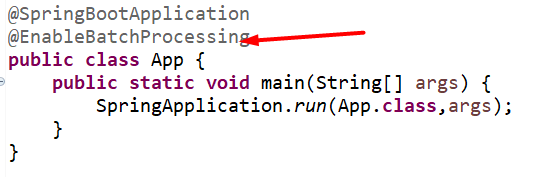


schema配置的是项目启动时创建数据表的SQL脚本，该脚本由Spring Batch提供；

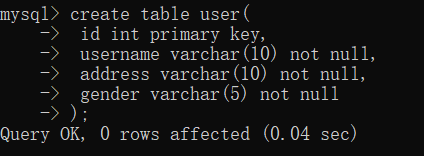
initialize-schema配置表示在项目启动时执行建表SQL；

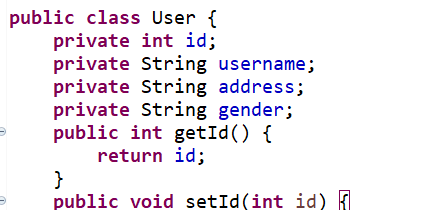
enabled表示禁止Spring Batch自动执行；默认情况下，当项目启动时就会执行配置好的批处理操作，禁止Spring Batch自动执行后，需要用户手动触发，例如发送一个请求。

1. 在项目启动类上添加@EnableBatchProcessing注解开启Spring Batch支持。如下：



1. 在数据库中建立user表，用于存放从data.csv中读取的数据，并为该表创建实体类User，如下：





1. 配置批处理，如下：





创建CsvBatchJobConfig类进行Spring Batch配置，在该类中，注入了JobBuilderFactory对象StepBuilderFactory对象以及DataSource对象，其中JobBuilderFactory对象用来构建Job，StepBuilderFactory对象用来构建Step，DataSource对象用来做持久化操作，这里使用的持久化方案是Spring-Jdbc；

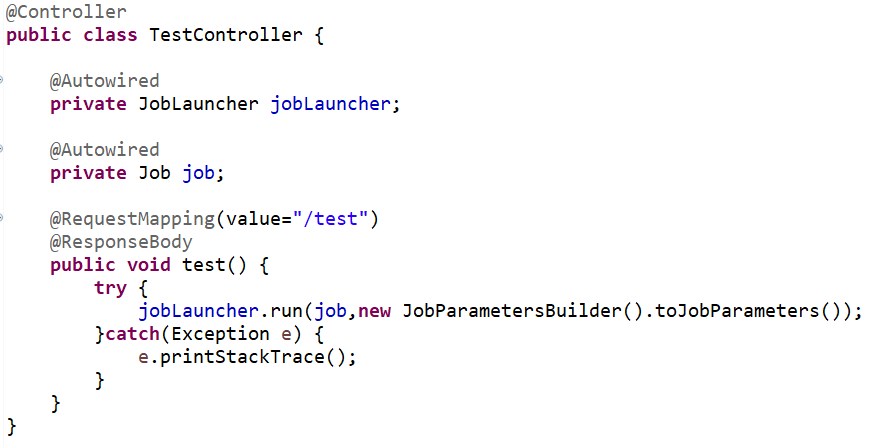
itemReader方法用来配置一个ItemReader，Spring Batch提供了一些常用的ItemReader，例如JdbcPagingItemReader用来读取数据库中的数据，StaxEventItemReader用来读取XML数据；在本例中使用的是FlatFileItemReader是用来加载普通的文件的ItemReader；在FlagFileItemReader的配置过程中，由于data.csv文件第一行是标题，因此通过setLineToSkip方法设置跳过一行，然后通过setResource方法配置data.csv文件的位置，本案例中csv文件存放在classpath目录下，然后通过setLineMapper方法设置每一行的数据信息；setNames方法配置了data.csv文件一共有4列，分别是id、username、address、gender；setDelimiter方法则是配置列与列之间的间隔符，通过间隔符对每一行的数据进行切分，最后设置要映射的实体类属性；

itemWriter方法表示数据的写出逻辑，Spring Batch也提供了多个ItemWriter的实现，常见的如FlatFileItemWriter，表示将数据写出为一个普通文件，StaxEventItemWriter表示将数据写出为XML；另外，还有针对不同数据库提供的写出操作支持类，如MongoItemWriter、JpaItemWriter、Neo4jItemWriter以及HibernateItemWriter等，本案例使用的JdbcBatchItemWriter则是通过JDBC将数据写出到一个关系型数据库中；JdbcBatchItemWriter主要配置数据以及数据插入SQL，主义占位符的写法是“:属性名”；最后通过BeanPropertyItemSqlParameterSourceProvider实例将实体类的数据和SQL中的占位符一一映射；

csvStep方法用于配置一个Step，通过stepBuilderFactory配置，首先通过get获取一个StepBuilder，get方法的参数就是该Step的name，然后调用chunk方法，参数2表示每读取到两条数据就执行一次writer操作，最后分别配置reader和writer；

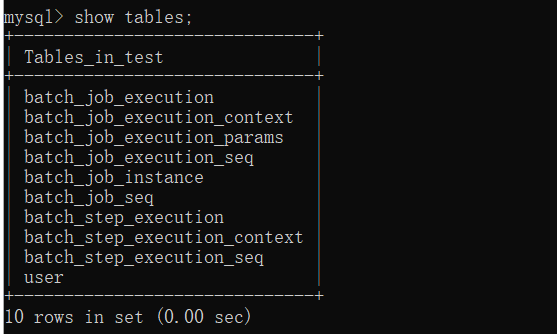
csvJob方法用于配置一个Job，通过jobBuilderFactory构建一个Job，get方法的参数为Job的name。

1. 编写控制器，进行测试，如下：



JobLauncher由框架提供，Job则是刚刚配置的；通过JobLauncher对象调用run方法启动一个批处理。

1. 启动项目，打开浏览器访问，访问/test路径，触发批处理，观察test库，如下：



数据库中会创建多个批处理相关的表，这些表用来记录批处理的执行状态，同时

data.csv中的数据也已经成功的写入了user表中，如下：



# Swagger2

## 简介

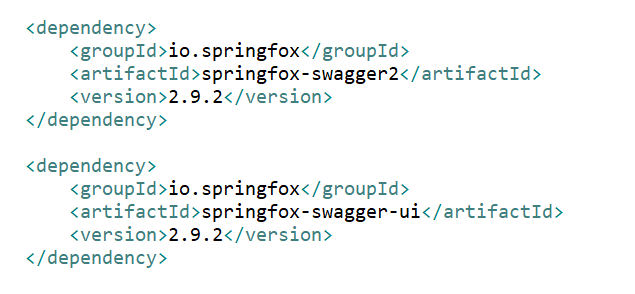
在前后端分离开发中，为了减少与其他团队的沟通成功，一般构建一份RESTful API文档来描述所有的接口信息，但是这种做法有很大的弊端，如：

1. 接口众多，编写RESTful API文档工作量巨大，因为RESTful API文档不仅要包含接口的基本信息，如接口地址、接口请求参数以及接口返回值等，还要包含HTTP请求类型、HTTP请求头、请求参数类型、返回值类型、所需权限等；
2. 维护不方便，一旦接口发生变化，就要修改文档；
3. 接口测试不方便，一般只能借助第三方工具（如Postman）来测试；

Swagger2是一个开源软件框架，可以帮助开发人员设计、构建、记录和使用RESTful Web服务，它将代码和文档融为一体，可以完美解决上面描述的问题，使开发人员将大部分精力集中到业务中，而不是繁琐的文档中。

## 整合Spring Boot

1. 添加依赖，如下：



1. 创建Swagger2配置类，如下：

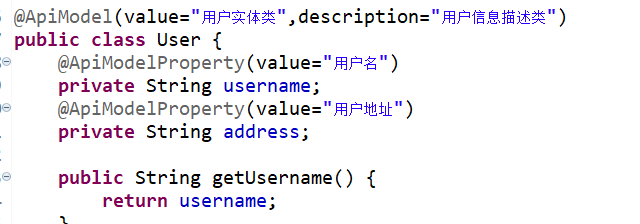


首先通过@EnableSwagger2注解开启Swagger2，然后最主要的是配置了一个Docket；关于Docket技术请查阅相关资料，因为它很重要；

通过apis方法配置要扫描的控制器位置，通过paths方法配置路径；

在apiInfo中构建文档的基本信息，例如描述、联系人信息、版本、标题等。

1. 创建实体类，这个实体是为了方便测试，并没有映射表，如下：



@ApiModel注解用于描述该实体的信息；

@ApiModelProperty注解用于描述实体类属性信息；

1. Swagger2配置完成后，开发接口（控制器），如下：





@Api注解标注在类上，用来描述整个@Controller信息；

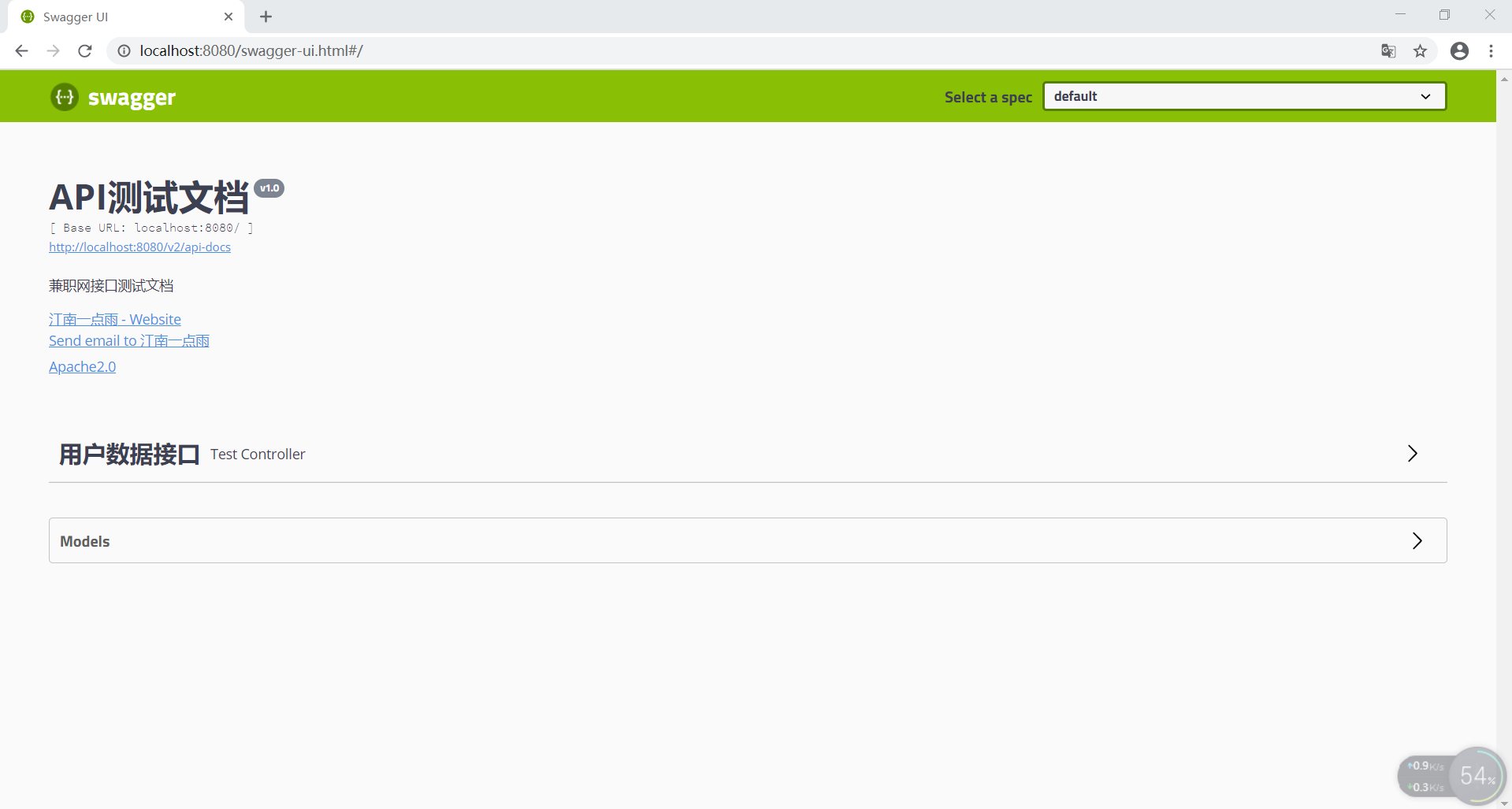
@ApiOperation注解标注在方法上，用来描述一个方法的基本信息，value属性是对方法作用的一个简短描述，notes属性则用来备注该方法的详细作用；

@ApiImplicitParam注解标注在方法上，用来描述方法的参数，paramType属性是指方法参数的类型，可选择值有path（参数获取方式@PathVariable）、query（参数获取方法@RequestParam）、header（参数获取方式@RequestHeader，用于将HTTP请求头中的属性绑定到形参中）、body以及form；name属性表示形参名，value属性是参数的描述信息；required属性表示该字段是否必填；defaultValue属性表示该字段的默认值；需要注意的是，这里的required属性defaultValue等字段只是文档上的约束描述，并不能真正的约束该接口（方法）；如果方法有多个参数，可以将多个参数的@ApiImplicitParam放到@ApilmpicitParams中；

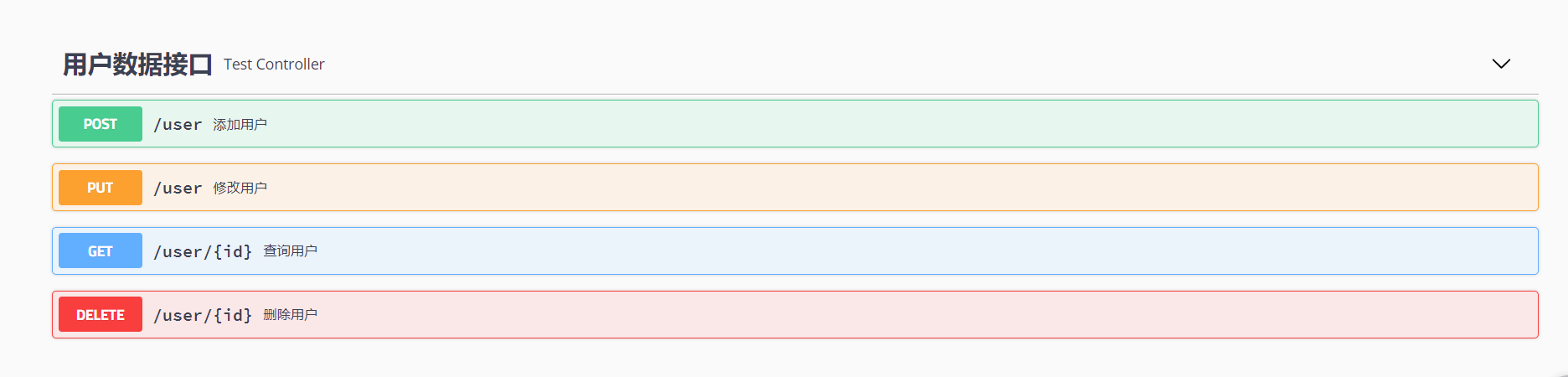
@ApiResponse注解是对响应结果的描述，code表示响应码，message表示相应的描述信息，若有多个@ApiResponse，则放在一个@ApiResponses注解中；

@ApiIgnore注解表示不对某个接口生成文档；

1. 启动项目，在浏览器中输入<http://localhost:8080/swagger-ui.html>即可看到接口文档，如下：



1. 展开用户数据接口，即可看到所有接口的描述，如下：



1. 展开一个接口描述，可以查看该接口的详细信息，单击Try it out按钮，可以实现对该接口的测试，如下：



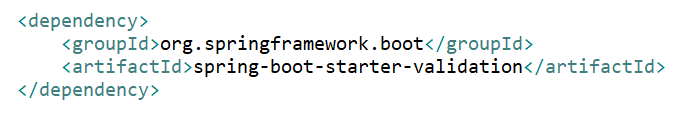
# 数据校验

在前端进行了数据校验并不意味着后端不用做数据校验，所以后端也是需要做数据校验的。对此Spring Boot对于后端的数据校验提供相关的自动化配置解决方案。

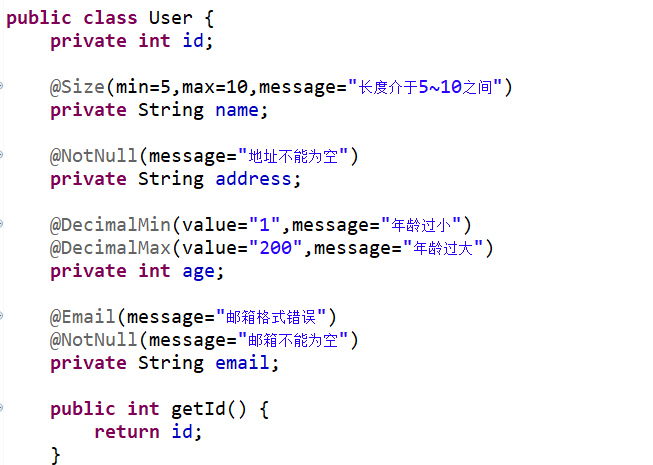
## 普通校验

普通校验是基础用法。

1. 添加spring-boot-starter-validation依赖，如下：



1. 创建User类，配置数据校验，如下：



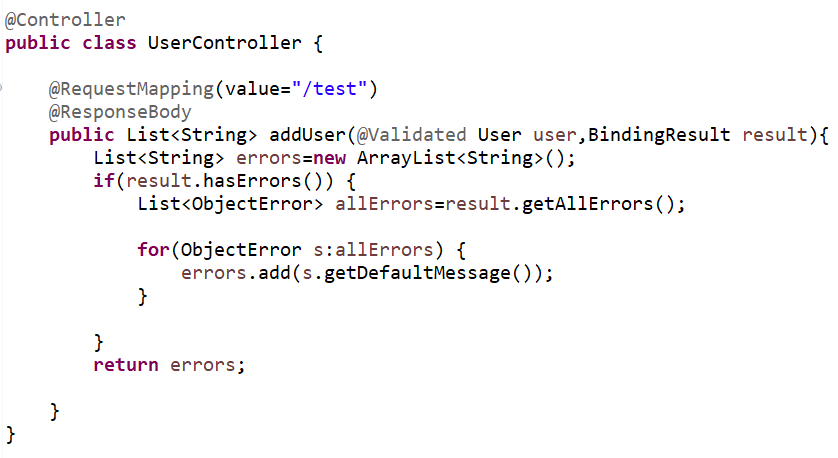
@Size注解表示一个字符串的长度或者一个集合的大小，必须在某一范围中，min属性指定范围的下限，max属性指定范围上限，message属性表示校验失败时的提示信息；

@NotNull注解表示该字段不能为空；

@DecimalMin注解表示取值下限，@DecimalMax注解表示取值上限；

@Email注解表示该属性要符合邮箱格式；

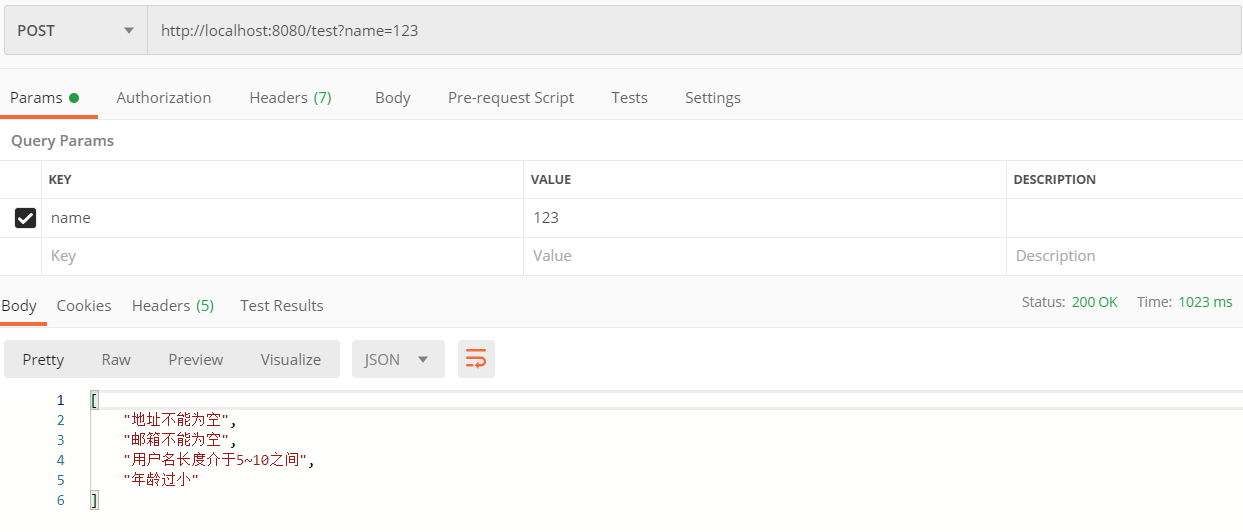
1. 创建控制器，如下：



形参User对象添加@Validated表示需要对该参数做校验；

形参BindingResult对象用于保存校验错误时的提示信息，如果该对象通过调用hasErrors方法发现校验有错误，则遍历取出，返回给前端；

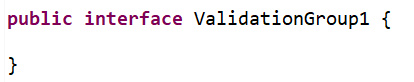
1. 启动项目，访问/test路径进行测试，如下：

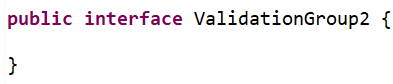


## 分组校验

开发者在实体类定义了很多校验规则，但是在某一业务中，并不需要对实体类中的每个属性都去进行校验，而是想对某个或某几个属性进行校验，此时可以使用分组校验完成。

1. 创建两个分组接口，如下：





1. 创建实体类User，为属性添加校验规则并分组，如下：



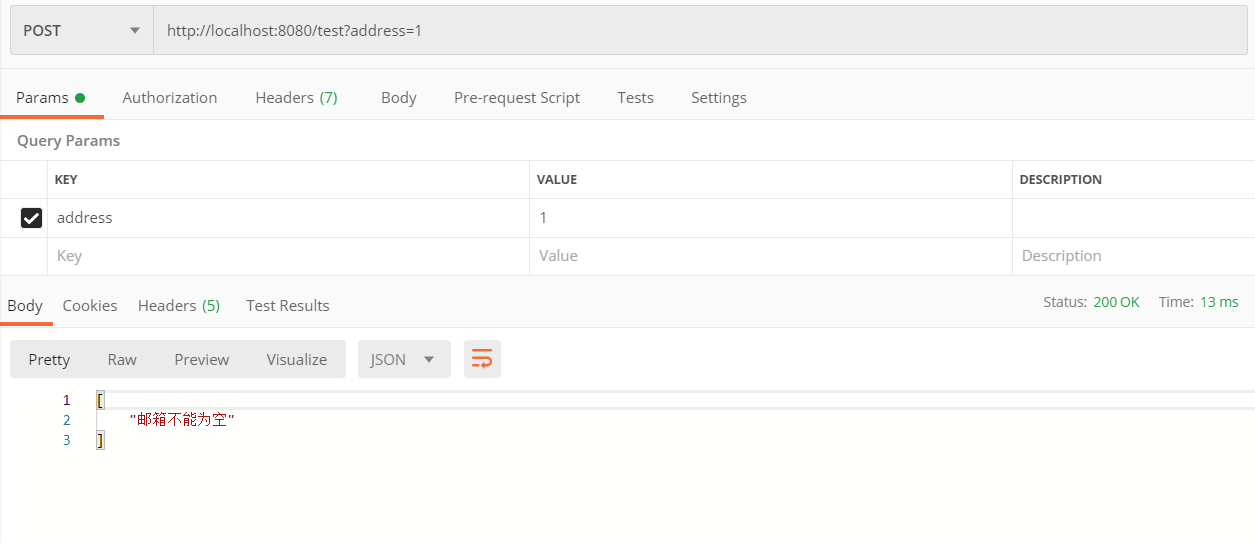
通过gourps属性进行分组；

1. 创建控制器，如下：



在使用@Validated注解标注要校验的参数时，同时还指定了校验规则所属的组；

1. 启动项目，进行测试，如下：



## 校验注解

除了上面介绍的几个常用的校验注解外，还有一些校验注解如下：

|  |  |  |
| --- | --- | --- |
| **校验注解** | **被标注的属性的类型** | **描述** |
| @AssertFalse | Boolean、boolean | 属性值必须为false |
| @AssertTrue | Boolean、boolean | 属性值必须为true |
| @DecimalMax | BigDecimal、BigInteger、CharSequence、byte、short、int、long以及各自的包装类 | 属性值必须小于或等于value属性指定的值 |
| @DecimalMin | BigDecimal、BigInteger、CharSequence、byte、short、int、long以及各自的包装类 | 属性值必须大于或等于value属性指定的值 |
| @Max | BigDecimal、BigInteger、byte、short、int、long以及它们各自的包装类 | 属性值必须小于或等于value属性指定的值 |
| @Min | BigDecimal、BigInteger、byte、short、int、long以及它们各自的包装类 | 属性值必须大于或等于value属性指定的值 |
| @Digits | BigDecimal、BigInteger、CharSequence、byte、short、int、long以及它们各自的包装类 | 属性值必须是一个数字，其值必须在可接受的范围内（整数位数和小数位数在指定范围内） |
| @Email | CharSequence | 属性值必须是Email格式 |
| @Future | java.util.Date、java.util.Calendar以及java.time包下与时间有关的类 | 属性值必须是一个未来的日期 |
| @Past | java.util.Date、java.util.Calendar以及java.time包下与时间有关的类 | 属性值必须是一个过去的日期 |
| @PastOrPresent | java.util.Date、java.util.Calendar以及java.time包下与时间有关的类 | 属性值必须是一个过去的日期或者是当前日期 |
| @FutureOrPresent | java.util.Date、java.util.Calender以及java.time包下与时间有关的类 | 属性值必须是一个未来的日期或者是当前日期 |
| @Negative | BigDecimal、BigInteger、byte、short、int、long以及它们各自的包装类 | 属性值必须是负数 |
| @NegativeOrZero | BigDecimal、BigInteger、byte、short、int、long以及它们各自的包装类 | 属性值必须是负数或0 |
| @Positive | BigDecimal、BigInteger、byte、short、int、long以及它们各自的包装类 | 属性值必须是正数 |
| @PositiveOrZero | BigDecimal、BigInteger、byte、short、int、long以及它们各自的包装类 | 属性值必须是正数或0 |
| @NotBlank | CharSequence | 属性值必须不为null、并且至少有一个非空白的字符 |
| @NotEmpty | CharSequence、Collection、Map、Array | 被注解的字符串的值必须不为null或空白字符串，被注解的集合或数组不能为空（与@NotBlank相比，一个空格字符串在@NotBlank中校验不通过，但是在NotEmpty中校验可通过） |
| @NotNull | 任意类型 | 属性值不能为null |
| @Null | 任意类型 | 属性值不能为null |
| @Pattern | CharSequence | 属性值必须符合指定的正则表达式 |
| @Size | CharSequence、Collection、Map、Array | 被注解的字符串属性、集合属性或者数组属性的长度必须在指定范围内，通过注解的min属性和max属性指定 |

CharSequence是一个描述字符串的接口，它有3个常用的子类，String、StringBuffer、StringBuilder。