Java EE 架构与应用

基于 SpringMVC 的体育健身网站设计报告

版本：3.0

小组成员：

覃依依 16301087

王佳乐 16301076

创建日期：

2019/5/13

1. 概述

该项目为基于 SpringMVC 架构的体育健身网站，较上一版本所增加的内容为：

• 使用 Kafka 作为消息块，将请求委托给后端请求处理器，并重新设计 REST API 以实现异步处理。

2. 总体设计

2.1 开发环境

该项目较上一版本所增加的开发环境及工具如下表所示，3.0 版本中详细的开发环境描述请参见《基于 SpringMVC 的体育健身网站设计报告》1.0 及 2.0 版本中 2.1 开发环境。

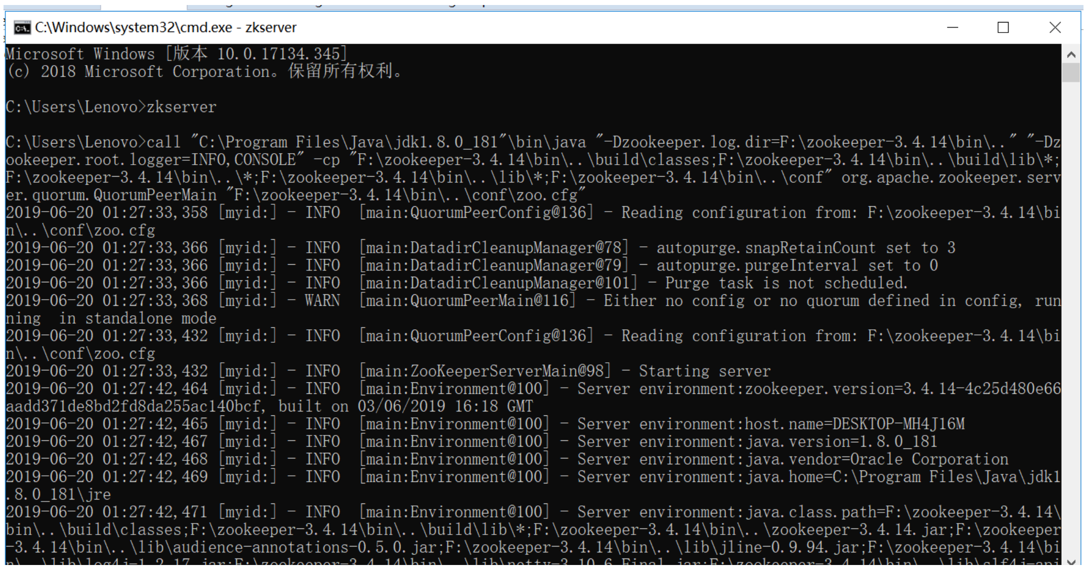
工具/环境	作用
kafka-test	分布式消息队列，以实现非阻塞 Rest 服务
ZooKeeper	用于协调服务器或集群拓扑，是配置信息的一致性文件系统，Kafka 使用 ZooKeeper 管理集群

表 2-1：新增的开发环境或工具

3. 详细设计

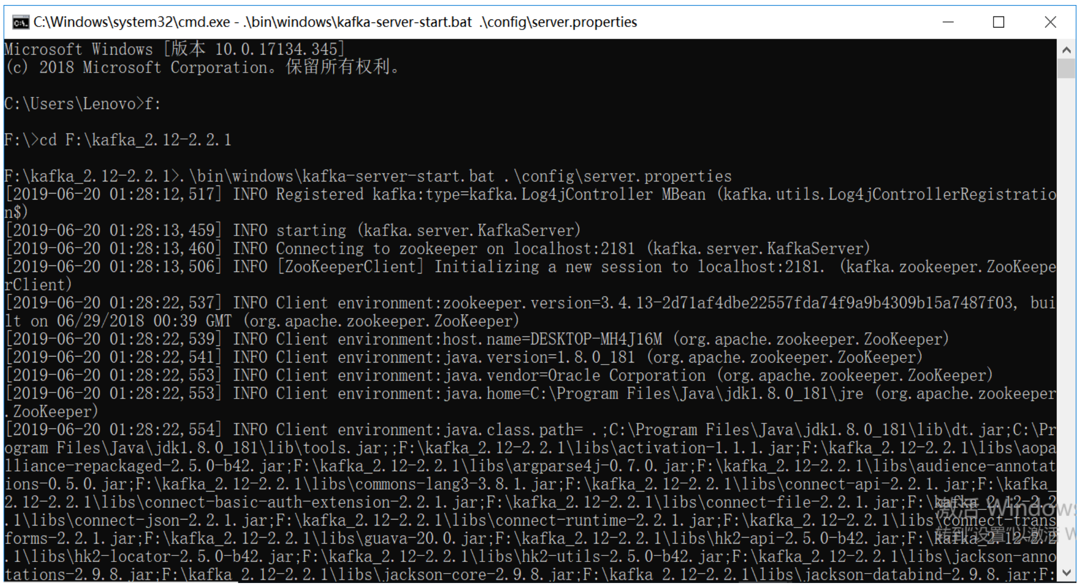
3.1 启动工作

Zookeeper 的启动与使用如下所示：



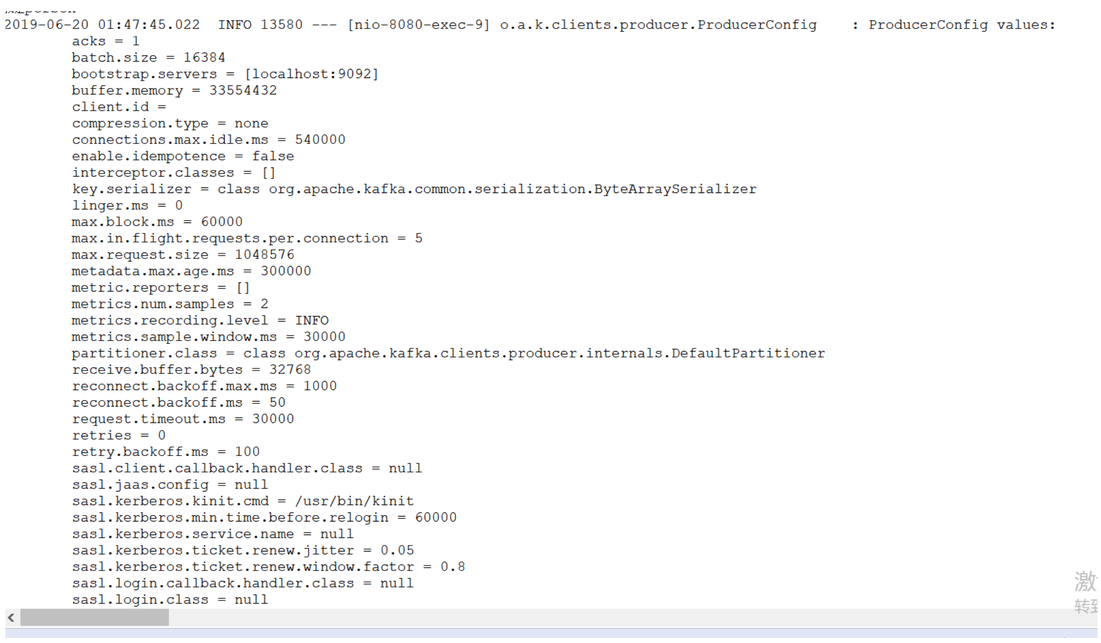
```
Microsoft Windows [版本 10.0.17134.345]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\Lenovo>zkserver

C:\Users\Lenovo>call "C:\Program Files\Java\jdk1.8.0_181"\bin\java "-Dzookeeper.log.dir=F:\zookeeper-3.4.14\bin\.." "-Dz
ookeeper.root.logger=INFO,CONSOLE" -cp "F:\zookeeper-3.4.14\bin\..\build\classes;F:\zookeeper-3.4.14\bin\..\build\lib\*;
F:\zookeeper-3.4.14\bin\..\*;F:\zookeeper-3.4.14\bin\..\lib\*;F:\zookeeper-3.4.14\bin\..\conf" org.apache.zookeeper.serv
er.quorum.QuorumPeerMain "F:\zookeeper-3.4.14\bin\..\conf\zoo.cfg"
2019-06-20 01:27:33,358 [myid:] - INFO  [main:QuorumPeerConfig@136] - Reading configuration from: F:\zookeeper-3.4.14\bi
n\..\conf\zoo.cfg
2019-06-20 01:27:33,366 [myid:] - INFO  [main:DatadirCleanupManager@78] - autopurge.snapRetainCount set to 3
2019-06-20 01:27:33,366 [myid:] - INFO  [main:DatadirCleanupManager@79] - autopurge.purgeInterval set to 0
2019-06-20 01:27:33,366 [myid:] - INFO  [main:DatadirCleanupManager@101] - Purge task is not scheduled.
2019-06-20 01:27:33,368 [myid:] - WARN  [main:QuorumPeerMain@116] - Either no config or no quorum defined in config, run
ning  in standalone mode
2019-06-20 01:27:33,432 [myid:] - INFO  [main:QuorumPeerConfig@136] - Reading configuration from: F:\zookeeper-3.4.14\bi
n\..\conf\zoo.cfg
2019-06-20 01:27:33,432 [myid:] - INFO  [main:ZooKeeperServerMain@98] - Starting server
2019-06-20 01:27:42,464 [myid:] - INFO  [main:Environment@100] - Server environment:zookeeper.version=3.4.14-4c25d480e66
aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
2019-06-20 01:27:42,465 [myid:] - INFO  [main:Environment@100] - Server environment:host.name=DESKTOP-MH4J16M
2019-06-20 01:27:42,467 [myid:] - INFO  [main:Environment@100] - Server environment:java.version=1.8.0_181
2019-06-20 01:27:42,468 [myid:] - INFO  [main:Environment@100] - Server environment:java.vendor=Oracle Corporation
2019-06-20 01:27:42,469 [myid:] - INFO  [main:Environment@100] - Server environment:java.home=C:\Program Files\Java\jdk1
.8.0_181\jre
2019-06-20 01:27:42,471 [myid:] - INFO  [main:Environment@100] - Server environment:java.class.path=F:\zookeeper-3.4.14\
bin\..\build\classes;F:\zookeeper-3.4.14\bin\..\build\lib\*;F:\zookeeper-3.4.14\bin\..\zookeeper-3.4.14.jar;F:\zookeeper
-3.4.14\bin\..\lib\audience-annotations-0.5.0.jar;F:\zookeeper-3.4.14\bin\..\lib\jline-0.9.94.jar;F:\zookeeper-3.4.14\bi
```

Kafka 的启动如下所示：



```
C:\Windows\system32\cmd.exe - .\bin\windows\kafka-server-start.bat  .\config\server.properties
Microsoft Windows [版本 10.0.17134.345]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\Lenovo>f:

F:\>cd F:\kafka_2.12-2.2.1

F:\kafka_2.12-2.2.1>.\bin\windows\kafka-server-start.bat .\config\server.properties
[2019-06-20 01:28:12,517] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistratio
n$)
[2019-06-20 01:28:13,459] INFO starting (kafka.server.KafkaServer)
[2019-06-20 01:28:13,460] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2019-06-20 01:28:13,506] INFO [ZooKeeperClient] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeepe
rClient)
[2019-06-20 01:28:22,537] INFO Client environment:zookeeper.version=3.4.13-2d71af4dbe22557fda74f9a9b4309b15a7487f03, bui
lt on 06/29/2018 00:39 GMT (org.apache.zookeeper.ZooKeeper)
[2019-06-20 01:28:22,539] INFO Client environment:host.name=DESKTOP-MH4J16M (org.apache.zookeeper.ZooKeeper)
[2019-06-20 01:28:22,541] INFO Client environment:java.version=1.8.0_181 (org.apache.zookeeper.ZooKeeper)
[2019-06-20 01:28:22,553] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2019-06-20 01:28:22,553] INFO Client environment:java.home=C:\Program Files\Java\jdk1.8.0_181\jre (org.apache.zookeeper
.ZooKeeper)
[2019-06-20 01:28:22,554] INFO Client environment:java.class.path= .;C:\Program Files\Java\jdk1.8.0_181\lib\dt.jar;C:\Pr
ogram Files\Java\jdk1.8.0_181\lib\tools.jar;;F:\kafka_2.12-2.2.1\libs\activation-1.1.1.jar;F:\kafka_2.12-2.2.1\libs\aopa
lliance-repackaged-2.5.0-b42.jar;F:\kafka_2.12-2.2.1\libs\argparse4j-0.7.0.jar;F:\kafka_2.12-2.2.1\libs\audience-annotat
ions-0.5.0.jar;F:\kafka_2.12-2.2.1\libs\commons-lang3-3.8.1.jar;F:\kafka_2.12-2.2.1\libs\connect-api-2.2.1.jar;F:\kafka_
2.12-2.2.1\libs\connect-basic-auth-extension-2.2.1.jar;F:\kafka_2.12-2.2.1\libs\connect-file-2.2.1.jar;F:\kafka_2.12-2.2
.1\libs\connect-json-2.2.1.jar;F:\kafka_2.12-2.2.1\libs\connect-runtime-2.2.1.jar;F:\kafka_2.12-2.2.1\libs\connect-trans
forms-2.2.1.jar;F:\kafka_2.12-2.2.1\libs\guava-20.0.jar;F:\kafka_2.12-2.2.1\libs\hk2-api-2.5.0-b42.jar;F:\kafka_2.12-2.2
.1\libs\hk2-locator-2.5.0-b42.jar;F:\kafka_2.12-2.2.1\libs\hk2-utils-2.5.0-b42.jar;F:\kafka_2.12-2.2.1\libs\jackson-anno
tations-2.9.8.jar;F:\kafka_2.12-2.2.1\libs\jackson-core-2.9.8.jar;F:\kafka_2.12-2.2.1\libs\jackson-databind-2.9.8.jar;F:
```

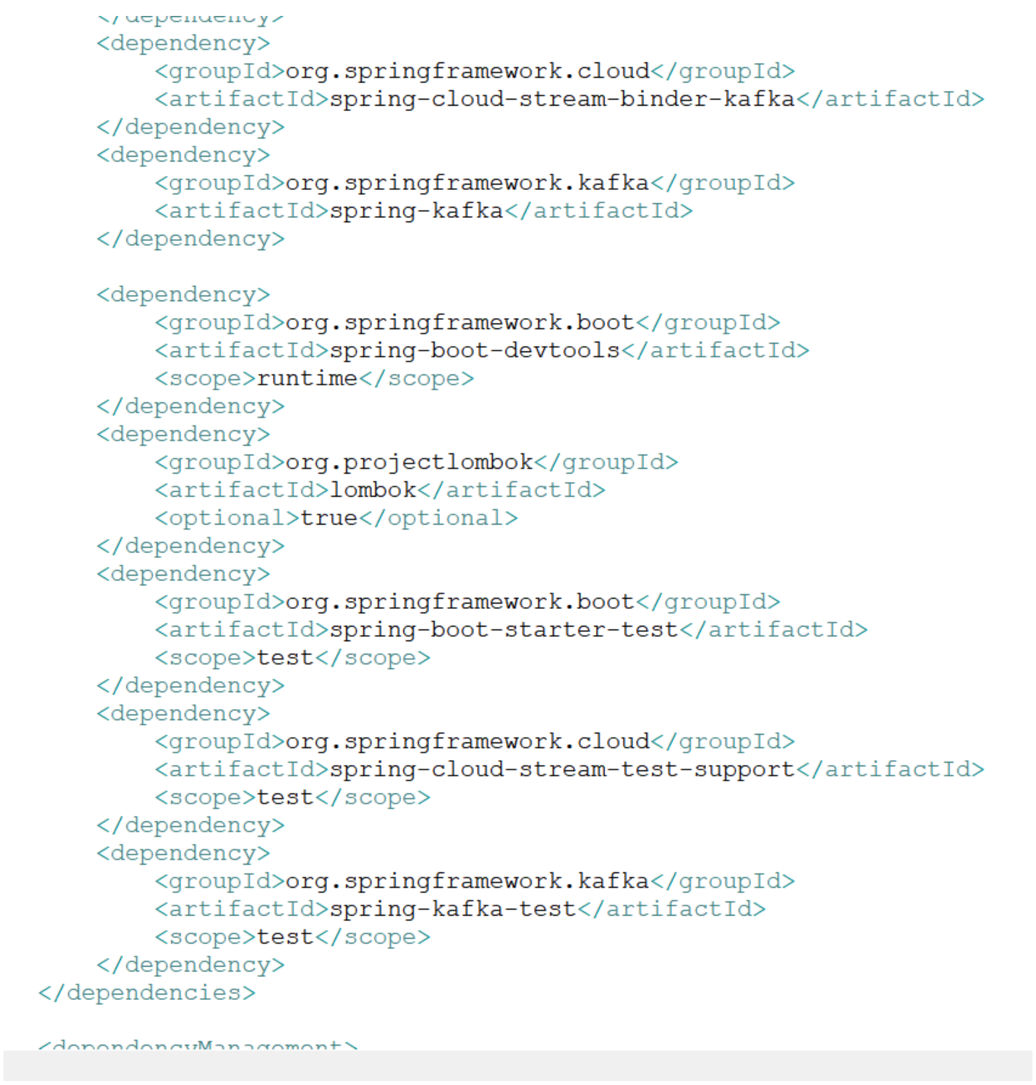
Springboot 与 Kafka 的连接如下所示：



```
2019-06-20 01:47:45.022  INFO 13580 --- [nio-8080-exec-9] o.a.k.clients.producer.ProducerConfig    : ProducerConfig values:
	acks = 1
	batch.size = 16384
	bootstrap.servers = [localhost:9092]
	buffer.memory = 33554432
	client.id =
	compression.type = none
	connections.max.idle.ms = 540000
	enable.idempotence = false
	interceptor.classes = []
	key.serializer = class org.apache.kafka.common.serialization.ByteArraySerializer
	linger.ms = 0
	max.block.ms = 60000
	max.in.flight.requests.per.connection = 5
	max.request.size = 1048576
	metadata.max.age.ms = 300000
	metric.reporters = []
	metrics.num.samples = 2
	metrics.recording.level = INFO
	metrics.sample.window.ms = 30000
	partitioner.class = class org.apache.kafka.clients.producer.internals.DefaultPartitioner
	receive.buffer.bytes = 32768
	reconnect.backoff.max.ms = 1000
	reconnect.backoff.ms = 50
	request.timeout.ms = 30000
	retries = 0
	retry.backoff.ms = 100
	sasl.client.callback.handler.class = null
	sasl.jaas.config = null
	sasl.kerberos.kinit.cmd = /usr/bin/kinit
	sasl.kerberos.min.time.before.relogin = 60000
	sasl.kerberos.service.name = null
	sasl.kerberos.ticket.renew.jitter = 0.05
	sasl.kerberos.ticket.renew.window.factor = 0.8
	sasl.login.callback.handler.class = null
	sasl.login.class = null
```

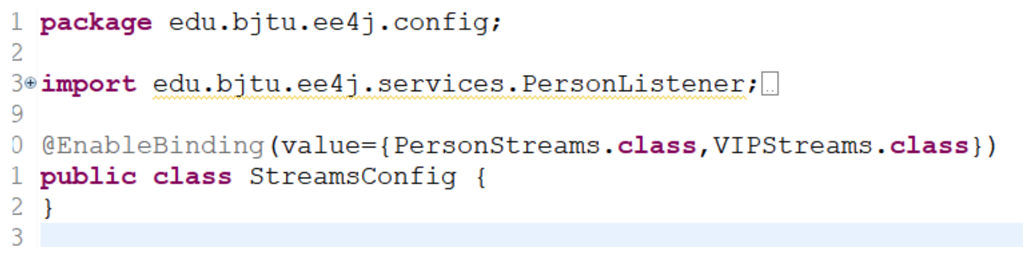
3.2 Springboot 中关于 kafka 的配置

引入相关依赖：



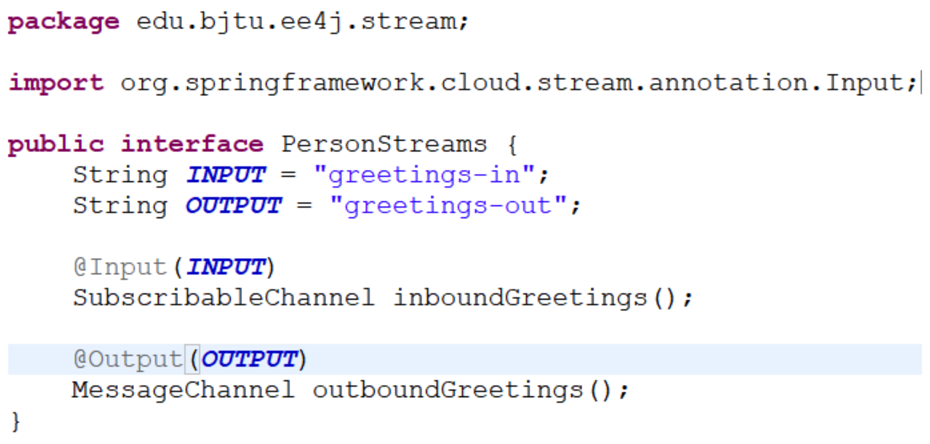
```xml
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-stream-binder-kafka</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-stream-test-support</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <dependencyManagement>
```

配置文件中绑定监听接口：

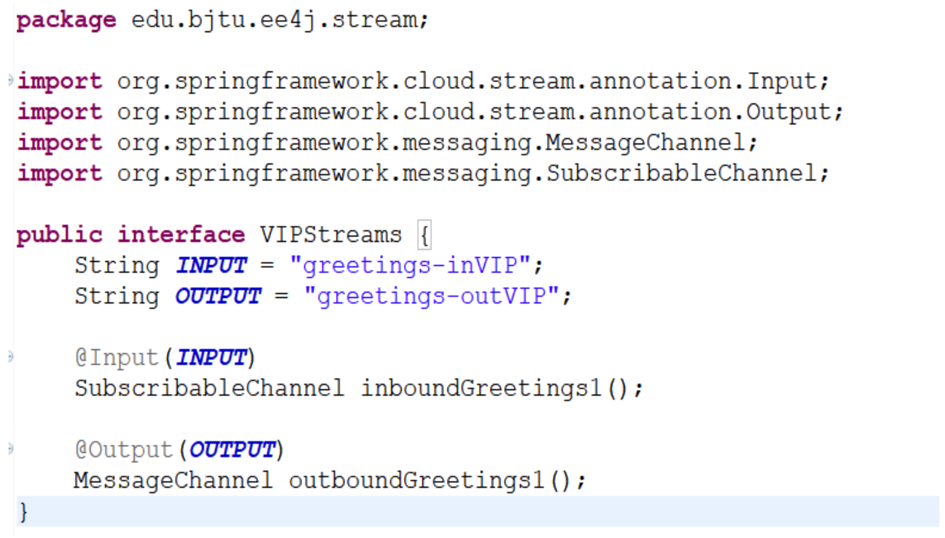


```java
package edu.bjtu.ee4j.config;

import edu.bjtu.ee4j.services.PersonListener;

@EnableBinding(value={PersonStreams.class,VIPStreams.class})
public class StreamsConfig {
}

```

接收与发送频道：



```java
package edu.bjtu.ee4j.stream;

import org.springframework.cloud.stream.annotation.Input;

public interface PersonStreams {
    String INPUT = "greetings-in";
    String OUTPUT = "greetings-out";

    @Input(INPUT)
    SubscribableChannel inboundGreetings();

    @Output(OUTPUT)
    MessageChannel outboundGreetings();
}
```

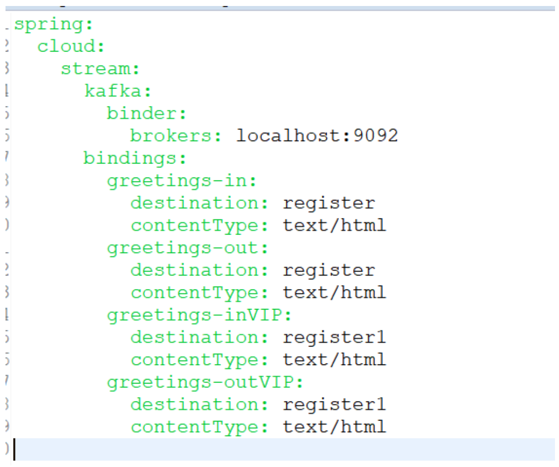


```java
package edu.bjtu.ee4j.stream;

import org.springframework.cloud.stream.annotation.Input;
import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;
import org.springframework.messaging.SubscribableChannel;

public interface VIPStreams {
    String INPUT = "greetings-inVIP";
    String OUTPUT = "greetings-outVIP";

    @Input(INPUT)
    SubscribableChannel inboundGreetings1();

    @Output(OUTPUT)
    MessageChannel outboundGreetings1();
}
```

3.3 功能实现

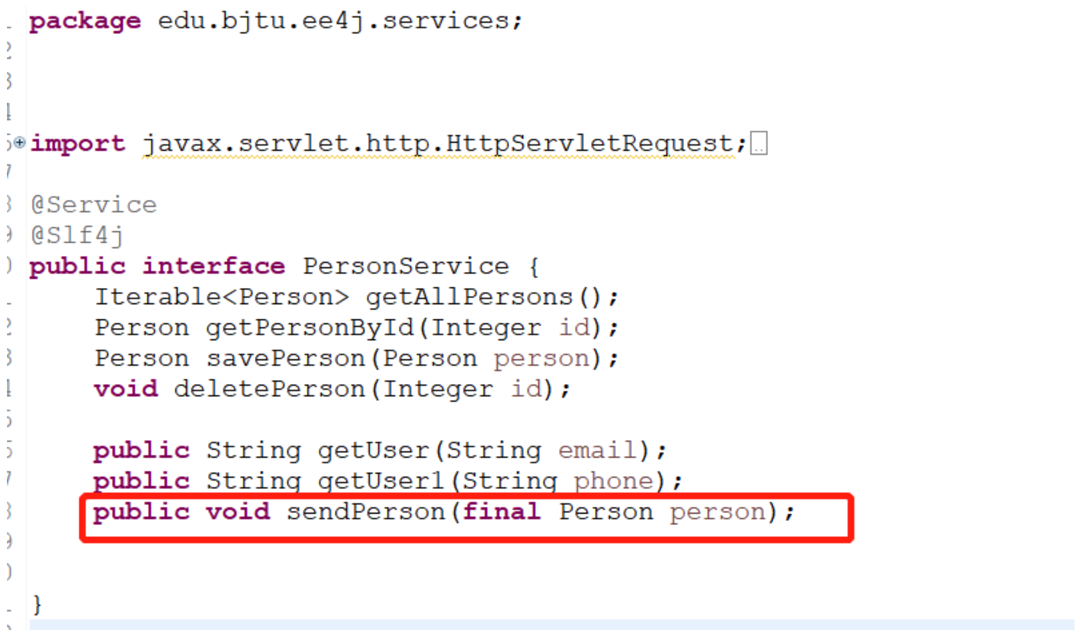
3.3.1 会员申请

Kafaka 的使用及相关代码截图:

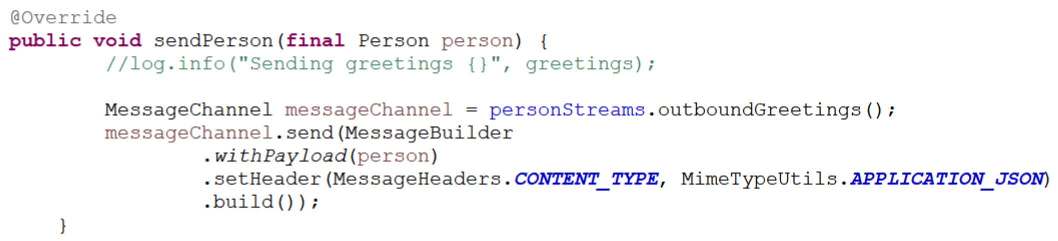


```
spring:
  cloud:
    stream:
      kafka:
        binder:
          brokers: localhost:9092
      bindings:
        greetings-in:
          destination: register
          contentType: text/html
        greetings-out:
          destination: register
          contentType: text/html
        greetings-inVIP:
          destination: register1
          contentType: text/html
        greetings-outVIP:
          destination: register1
          contentType: text/html
```

这里配置了两个 topic,因为要完成对两个事件的处理（会员申请和私教课申请），两个 topic 起了不同的名字，让两个 consumer 分别进行处理，否则 consumer 会对 producer 产生的广播消息都进行处理，但是要访问两个不同的数据库，所以用了两个 topic，放置程序报错。

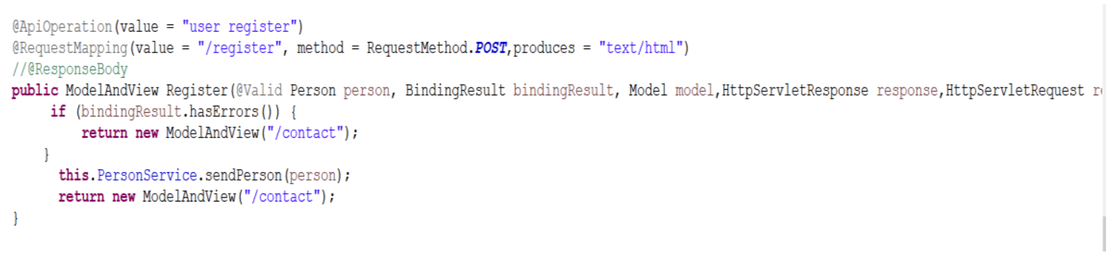


```
package edu.bjtu.ee4j.services;



import javax.servlet.http.HttpServletRequest;

@Service
@Slf4j
public interface PersonService {
    Iterable<Person> getAllPersons();
    Person getPersonById(Integer id);
    Person savePerson(Person person);
    void deletePerson(Integer id);

    public String getUser(String email);
    public String getUser1(String phone);
    public void sendPerson(final Person person);


}
```



```
@Override
public void sendPerson(final Person person) {
        //log.info("Sending greetings {}", greetings);

        MessageChannel messageChannel = personStreams.outboundGreetings();
        messageChannel.send(MessageBuilder
                .withPayload(person)
                .setHeader(MessageHeaders.CONTENT_TYPE, MimeTypeUtils.APPLICATION_JSON)
                .build());
    }
```

以上两图为申请成为会员时候的生产者，上面是 interface,下面是具体的实现，它发送消息给消费者，消费者进行处理，防止过多 post 请求高并发时，服务器崩溃。



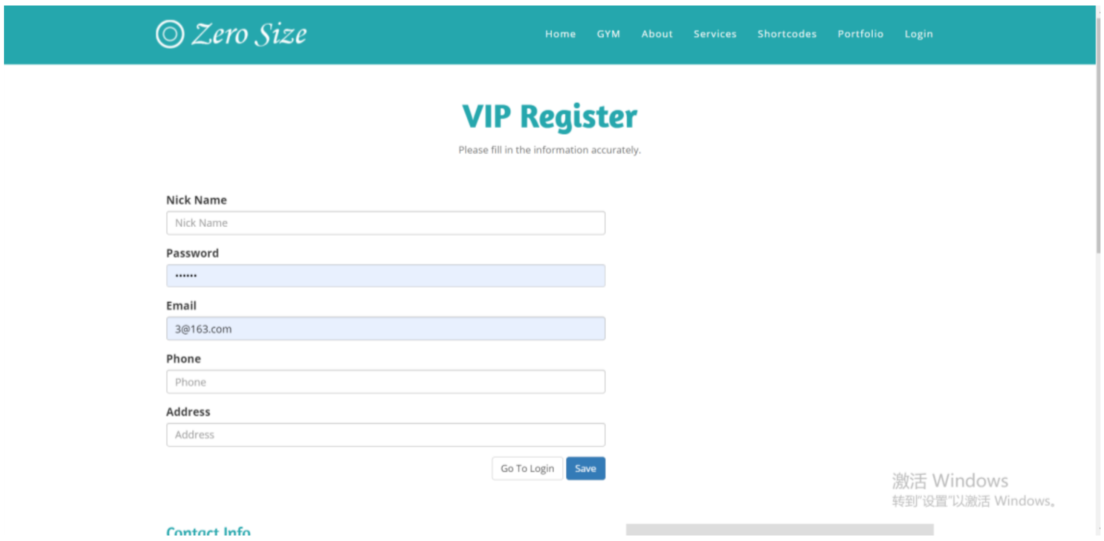
```
@ApiOperation(value = "user register")
@RequestMapping(value = "/register", method = RequestMethod.POST,produces = "text/html")
//@ResponseBody
public ModelAndView Register(@Valid Person person, BindingResult bindingResult, Model model,HttpServletResponse response,HttpServletRequest r
     if (bindingResult.hasErrors()) {
         return new ModelAndView("/contact");
    }
      this.PersonService.sendPerson(person);
      return new ModelAndView("/contact");
}
```

上图为在 controller 中进行调用。

```
import edu.bjtu.ee4j.domain.Person;

@Component
@Slf4j
public class PersonListener {

    private PersonService PersonService;
    private VIPService vipService;

    @Autowired
    public void setPersonService(PersonService PersonService,VIPService vipService) {
        this.PersonService = PersonService;
        this.vipService = vipService;
    }

    @StreamListener(PersonStreams.INPUT)

    public void handleGreetings(Person person) {

        //Logger LOG = LoggerFactory.getLogger("SpringKafkaStream");
        //LOG.info("Received greetings: {}", person);
        boolean judge=true;
          System.out.println("我是person");


      if(this.PersonService.getUser(person.getEmail())!=null){
          //model.addAttribute("err", "The email has been registered!");
          judge=false;
    //    return new ModelAndView("/contact");
      }
      else if(this.PersonService.getUser1(person.getPhone_no())!=null){
         // model.addAttribute("err1", "The phone has been registered!");
          judge=false;
        //  return new ModelAndView("/contact");
      }

        this.PersonService.savePerson(person);
```

上图是申请会员事件消息的 listener（consumer）,对传过来的会员信息进行验证，通过验证的存入数据库中。

申请注册成为会员界面图：



VIP Register

Please fill in the information accurately.

Nick Name: Nick Name

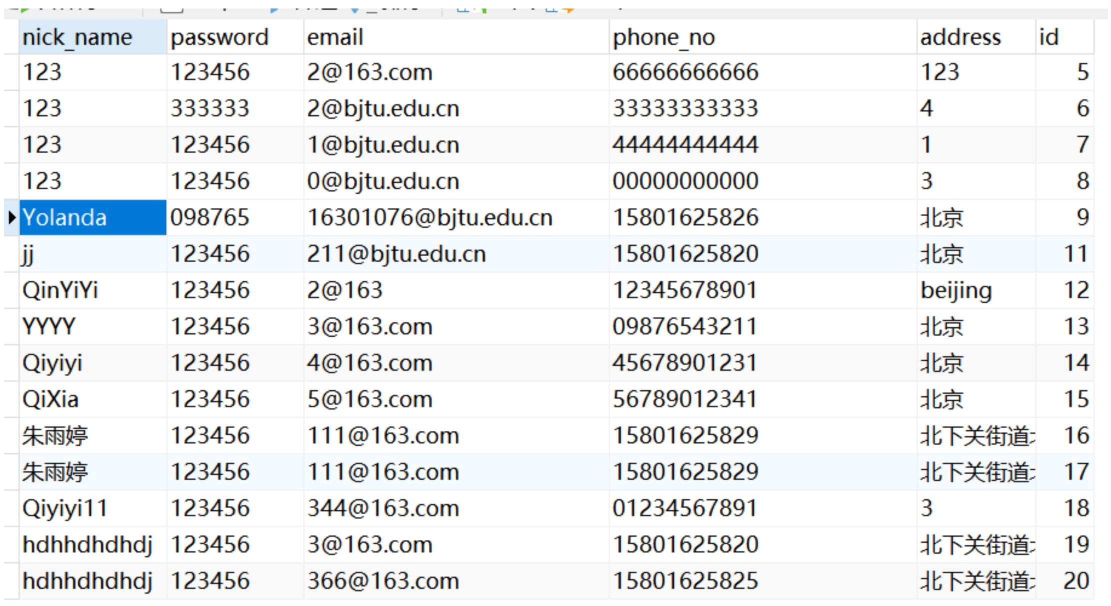
Password: ••••••

Email: 3@163.com

Phone: Phone

Address: Address

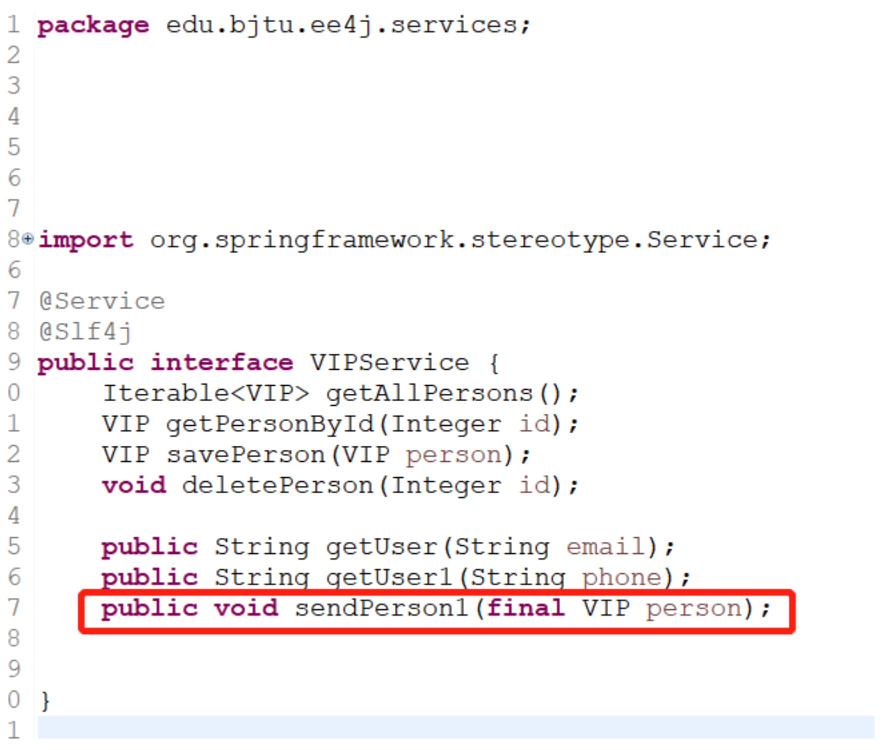
5.3 数据库中会员的注册记录



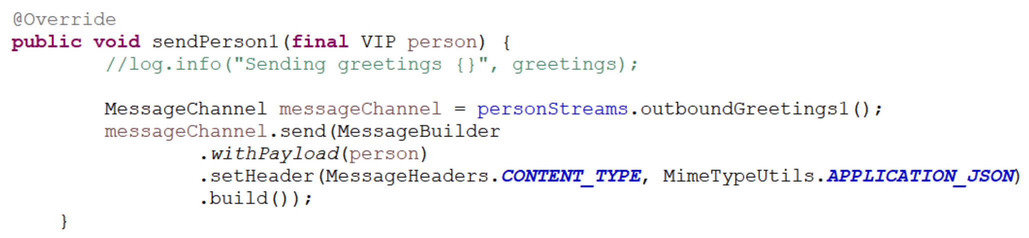
nick_name	password	email	phone_no	address	id
123	123456	2@163.com	66666666666	123	5
123	333333	2@bjtu.edu.cn	33333333333	4	6
123	123456	1@bjtu.edu.cn	44444444444	1	7
123	123456	0@bjtu.edu.cn	00000000000	3	8
Yolanda	098765	16301076@bjtu.edu.cn	15801625826	北京	9
jj	123456	211@bjtu.edu.cn	15801625820	北京	11
QinYiYi	123456	2@163	12345678901	beijing	12
YYYY	123456	3@163.com	09876543211	北京	13
Qiyiyi	123456	4@163.com	45678901231	北京	14
QiXia	123456	5@163.com	56789012341	北京	15
朱雨婷	123456	111@163.com	15801625829	北下关街道:	16
朱雨婷	123456	111@163.com	15801625829	北下关街道:	17
Qiyiyi11	123456	344@163.com	01234567891	3	18
hdhhdhdhdj	123456	3@163.com	15801625820	北下关街道:	19
hdhhdhdhdj	123456	366@163.com	15801625825	北下关街道:	20

3.3.2 会员申请私教课

kafaka 的使用及相关代码截图：

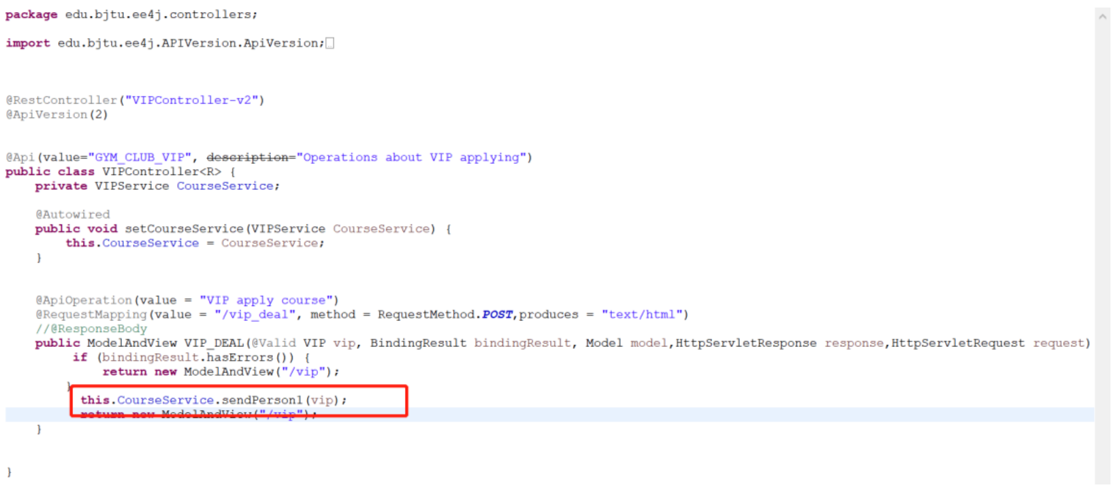


```
package edu.bjtu.ee4j.services;






import org.springframework.stereotype.Service;

@Service
@Slf4j
public interface VIPService {
    Iterable<VIP> getAllPersons();
    VIP getPersonById(Integer id);
    VIP savePerson(VIP person);
    void deletePerson(Integer id);

    public String getUser(String email);
    public String getUser1(String phone);
    public void sendPerson1(final VIP person);


}
```



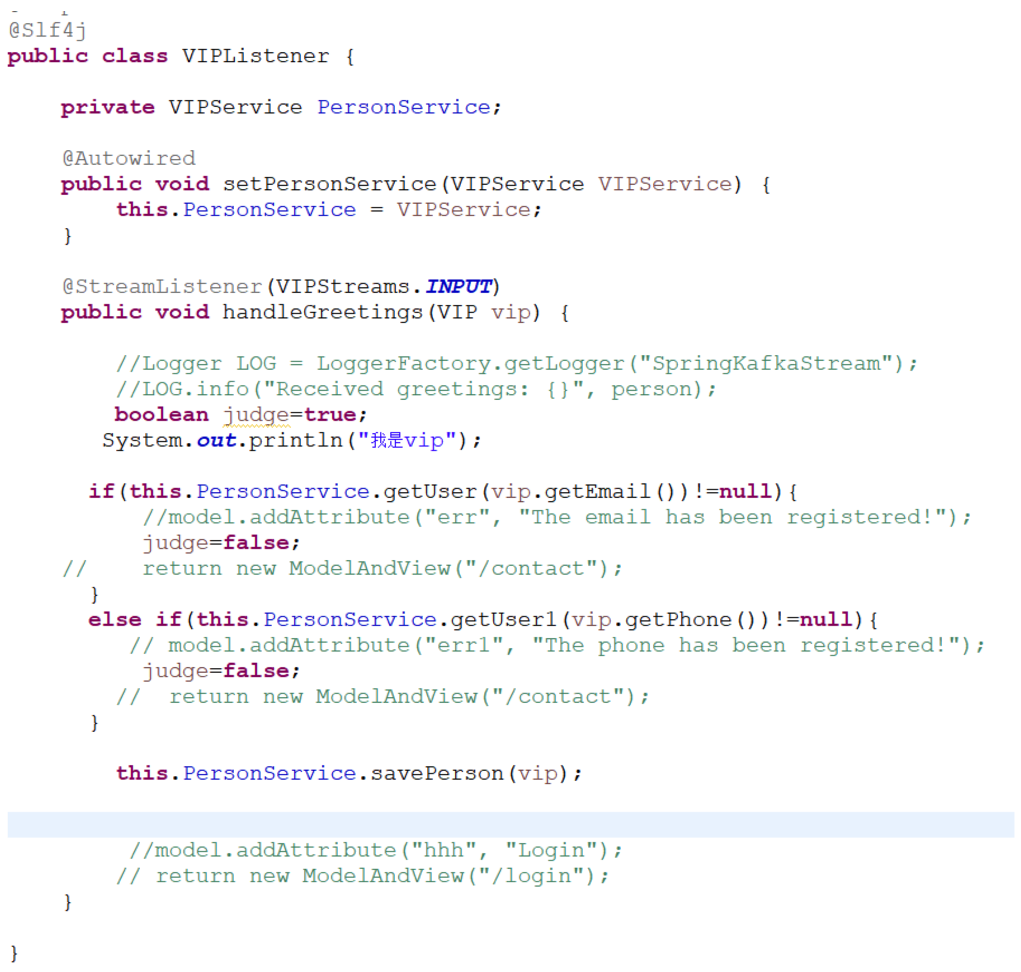
```
@Override
public void sendPerson1(final VIP person) {
        //log.info("Sending greetings {}", greetings);

        MessageChannel messageChannel = personStreams.outboundGreetings1();
        messageChannel.send(MessageBuilder
                .withPayload(person)
                .setHeader(MessageHeaders.CONTENT_TYPE, MimeTypeUtils.APPLICATION_JSON)
                .build());
    }
```

私教课申请的生产者，第一个图为 interface，第二个图为具体的实现。



```
package edu.bjtu.ee4j.controllers;

import edu.bjtu.ee4j.APIVersion.ApiVersion;


@RestController("VIPController-v2")
@ApiVersion(2)


@Api(value="GYM_CLUB_VIP", ~~description~~="Operations about VIP applying")
public class VIPController<R> {
    private VIPService CourseService;

    @Autowired
    public void setCourseService(VIPService CourseService) {
        this.CourseService = CourseService;
    }


    @ApiOperation(value = "VIP apply course")
    @RequestMapping(value = "/vip_deal", method = RequestMethod.POST,produces = "text/html")
    //@ResponseBody
    public ModelAndView VIP_DEAL(@Valid VIP vip, BindingResult bindingResult, Model model,HttpServletResponse response,HttpServletRequest request)
         if (bindingResult.hasErrors()) {
             return new ModelAndView("/vip");
         }
         this.CourseService.sendPerson1(vip);
         ~~return new ModelAndView("/vip");~~
    }


}
```

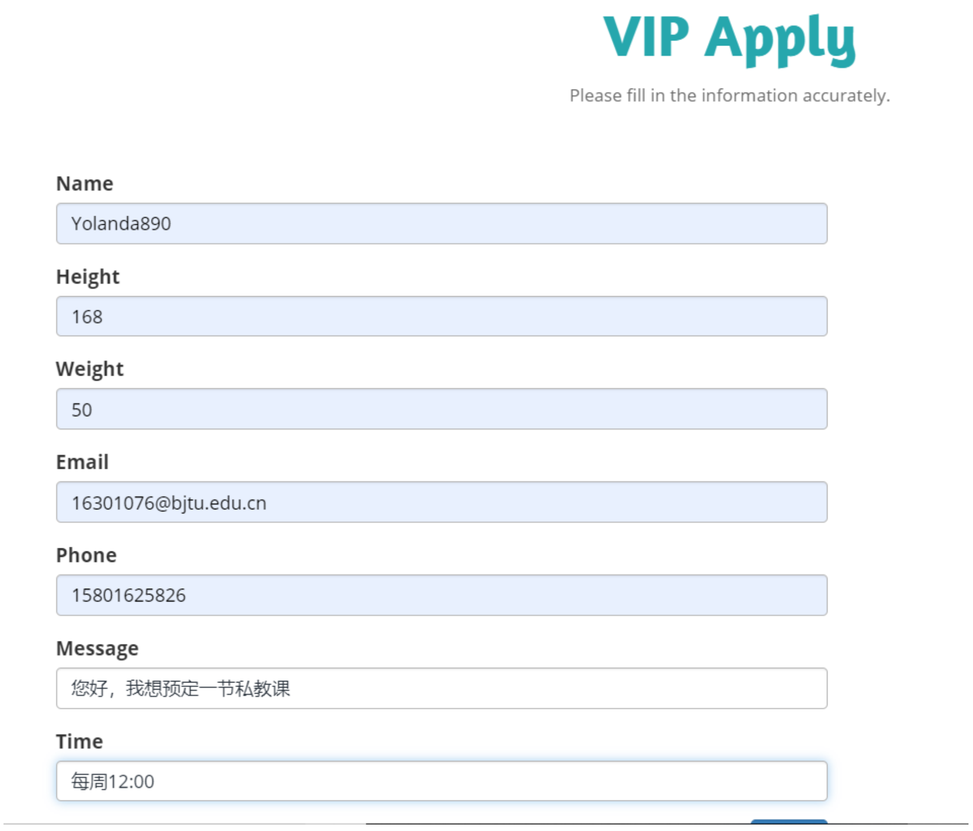
上图为在 controller 中进行调用。



```
@Slf4j
public class VIPListener {

    private VIPService PersonService;

    @Autowired
    public void setPersonService(VIPService VIPService) {
        this.PersonService = VIPService;
    }

    @StreamListener(VIPStreams.INPUT)
    public void handleGreetings(VIP vip) {

        //Logger LOG = LoggerFactory.getLogger("SpringKafkaStream");
        //LOG.info("Received greetings: {}", person);
        boolean judge=true;
       System.out.println("我是vip");

      if(this.PersonService.getUser(vip.getEmail())!=null){
          //model.addAttribute("err", "The email has been registered!");
          judge=false;
    //    return new ModelAndView("/contact");
      }
      else if(this.PersonService.getUser1(vip.getPhone())!=null){
         // model.addAttribute("err1", "The phone has been registered!");
          judge=false;
        //  return new ModelAndView("/contact");
      }

        this.PersonService.savePerson(vip);


         //model.addAttribute("hhh", "Login");
        // return new ModelAndView("/login");
    }

}
```

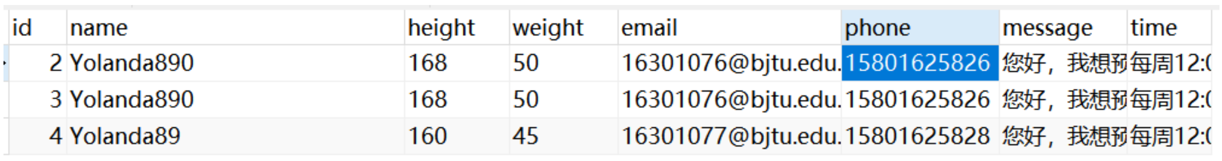
上图为申请私教课的 listener，也就是 consumer。

会员申请私教课的表单截图：



Name	Yolanda890
Height	168
Weight	50
Email	16301076@bjtu.edu.cn
Phone	15801625826
Message	您好，我想预定一节私教课
Time	每周12:00

数据库中会员对私教课的申请记录：



id	name	height	weight	email	phone	message	time
2	Yolanda890	168	50	16301076@bjtu.edu.	15801625826	您好，我想预	每周12:(
3	Yolanda890	168	50	16301076@bjtu.edu.	15801625826	您好，我想预	每周12:(
4	Yolanda89	160	45	16301077@bjtu.edu.	15801625828	您好，我想预	每周12:(