



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：

```

C:\Windows\system32\cmd.exe - zkserver
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\.. -Dzookeeper.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\..\lib\*;F:\zookeeper-3.4.14\bin\..\conf" org.apache.zookeeper.server.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\bin\..\conf\zoo.cfg
2019-06-20 01:27:33,366 [myid:] - INFO [main:DatadirCleanupManager@78] - autotune.snapRetainCount set to 3
2019-06-20 01:27:33,366 [myid:] - INFO [main:DatadirCleanupManager@79] - autotune.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, running in standalone mode
2019-06-20 01:27:33,432 [myid:] - INFO [main:QuorumPeerConfig@136] - Reading configuration from: F:\zookeeper-3.4.14\bin\..\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=3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, 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\bin\..\lib\log4j-1.2.17.jar;F:\zookeeper-3.4.14\bin\..\lib\slf4j-api-1.7.25.jar;F:\zookeeper-3.4.14\bin\..\lib\zookeeper-3.4.14.jar;F:\zookeeper-3.4.14\bin\..\lib\zookeeper-jute-3.4.14.jar;F:\zookeeper-3.4.14\bin\..\lib\zookeeper-3.4.14.jar

```

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.Log4jControllerRegistration$)  
[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.ZooKeeperClient)  
[2019-06-20 01:28:22,537] INFO Client environment:zookeeper.version=3.4.13-2d71af4d4be22557fda74f9a9b4309b15a7487f03, build.timestamp=2018-06-29T00:39 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:\Program 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\acorn-1.1.1.jar; F:\kafka_2.12-2.2.1\libs\antlr-3.1.3.jar; F:\kafka_2.12-2.2.1\libs\asm-6.0.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-transforms-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-launcher-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\json-schema-validator-2.9.8.jar; F:\kafka_2.12-2.2.1\libs\json-schema-core-2.9.8.jar; F:\kafka_2.12-2.2.1\libs\json-schema-databind-2.9.8.jar; F:
```

Springboot 与 Kafka 的连接如下所示:

```
2019-06-20 01:47:45.022 INFO 13800 --- [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:

```

</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:

```

1 package edu.bjtu.ee4j.config;
2
3 import edu.bjtu.ee4j.services.PersonListener;
4
5 @EnableBinding(value={PersonStreams.class,VIPStreams.class})
6 public class StreamsConfig {
7 }
8
9

```

接收与发送频道:

```
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();
}

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 request) {
    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

Go To Login Save

激活 Windows
转到“设置”以激活 Windows。

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：


```

1 package edu.bjtu.ee4j.services;
2
3
4
5
6
7
8+import org.springframework.stereotype.Service;
6
7 @Service
8 @Slf4j
9 public interface VIPService {
10     Iterable<VIP> getAllPersons();
11     VIP getPersonById(Integer id);
12     VIP savePerson(VIP person);
13     void deletePerson(Integer id);
14
15     public String getUser(String email);
16     public String getUser1(String phone);
17     public void sendPerson1(final VIP person);
18
19 }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

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：

VIP Apply

Please fill in the information accurately.

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12:00
3	Yolanda890	168	50	16301076@bjtu.edu.	15801625826	您好, 我想预	每周12:00
4	Yolanda89	160	45	16301077@bjtu.edu.	15801625828	您好, 我想预	每周12:00