

测试文档

一、接口测试

1、“我的”模块（负责人:221801239 林龙星）

1.1 根据 token 获取用户信息接口

请求方法和路径：Post /121.5.100.116:8080/user/getDetail

请求参数:

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIyIiwiaWF0IjoxNjM1NjAyLCJzdzWIIOi
Key	Value

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1
2      "msg": "操作成功",
3      "code": 200,
4      "data": {
5          "userSex": "男",
6          "userPic": "sss",
7          "userAccount": "15260011385",
8          "userNickname": "金",
9          "userId": 2,
10         "userProfile": "ssds"
11     }
12
```

1.2 根据 id 获取用户信息

请求方法和路径：Post /121.5.100.116:8080/user/getIntro

请求参数:

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIyIiwiaWF0IjoxNjM1NjAyLCJzdzWIIOi
Key	Value

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2      "userId": 2
3
4
```

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2    "msg": "操作成功",
3    "code": 200,
4    "data": {
5      "userSex": "男",
6      "userState": 0,
7      "userPic": "sss",
8      "userNickname": "金",
9      "userProfile": "sss"
10   }
11 }
```

1.3 关注/取消关注用户

请求方法和路径: Post /121.5.100.116:8080/user/subscribe

请求参数:

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZW4iLCJ0eXN0IjwzZjZdWllOi
Key	Value

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {
2    "userId": 3
3  }
4  }
```

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

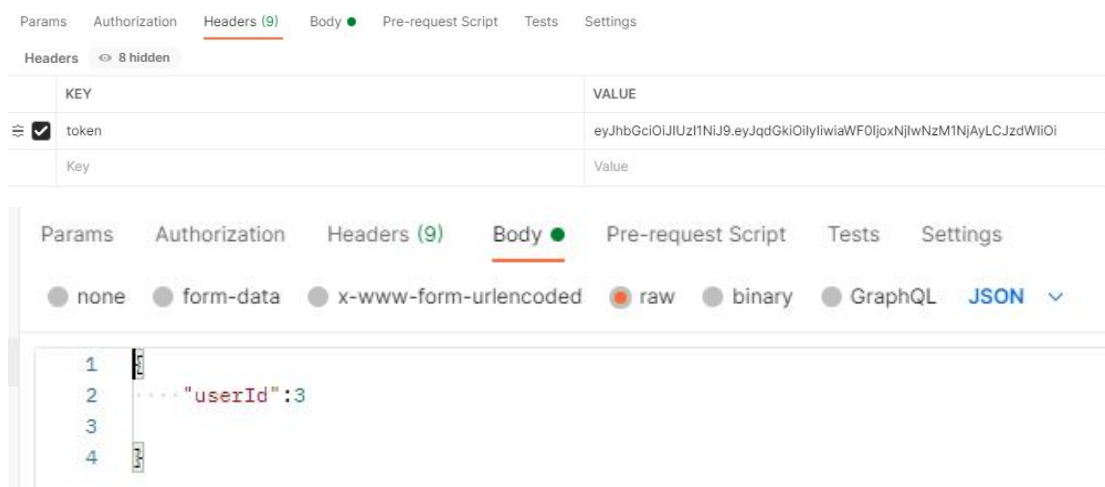
```
1  {
2    "msg": "关注成功",
3    "code": 200,
4    "data": null
5  }
```



1.4 获取用户粉丝

请求方法和路径: Post /121.5.100.116:8080/user/getSubscriber

请求参数:



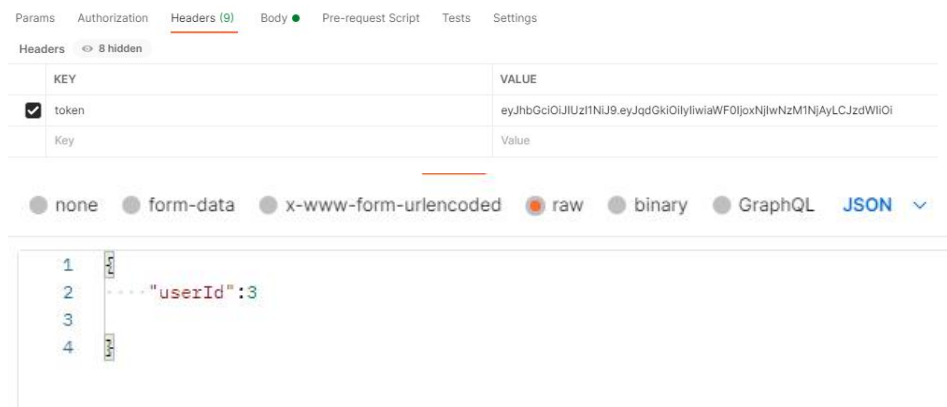
返回参数:



1.5 获取用户关注

请求方法和路径: Post/121.5.100.116:8080/user/getSubscription

请求参数:



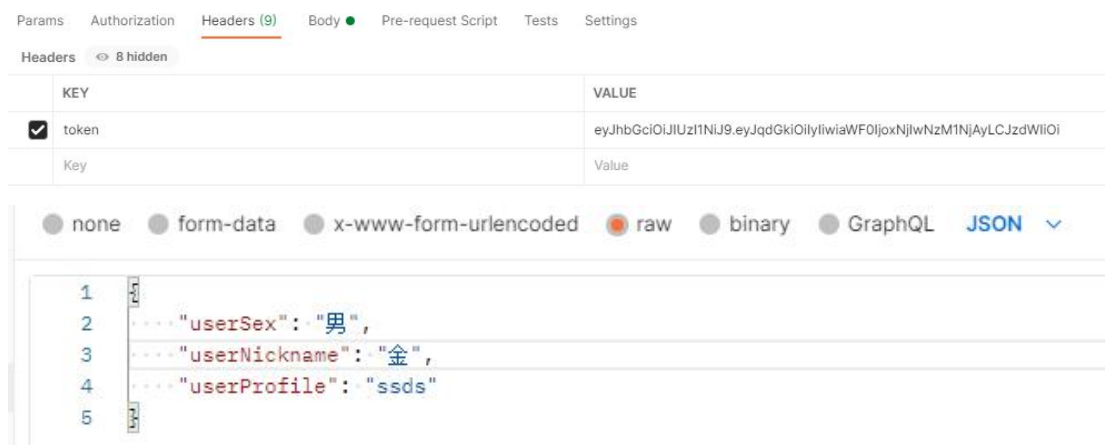
返回参数:



1.6 修改用户信息

请求方法和路径: Post/121.5.100.116:8080/user/modifyInfo

请求参数:



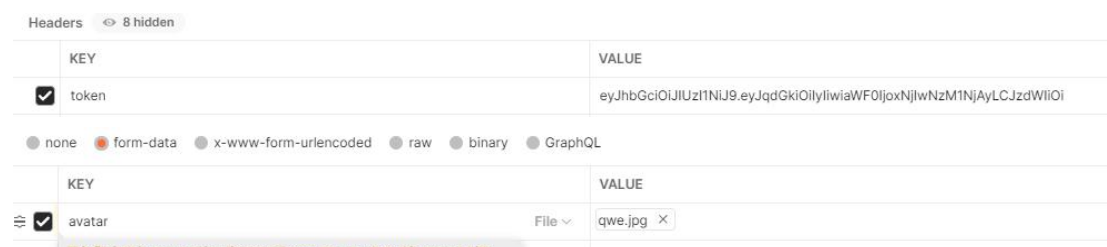
返回参数:



1.8 上传用户头像接口

请求方法和路径：Post/ 121.5.100.116:8080/user/uploadPic

请求参数：



返回参数：



2、“衣柜”模块（负责人:吴晗杰）

2.1 用户导入衣物

请求方法和路径：Post/121.5.100.116:8080/importClothing

请求参数：

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI2IiwiaWF0IjoxNjIwODM4MjExLCJzdWIiOi

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> subtypeld	2
<input checked="" type="checkbox"/> clothingPic	11.jpg <input type="button" value="x"/>

返回参数：

Body	Cookies	Headers (8)	Test Results
Pretty	Raw	Preview	Visualize
JSON			
1			
2	"msg": "操作成功",		
3	"code": 200,		
4	"data": null		
5			

2.2 获取某类别下的衣物

请求方法和路径：Post/121.5.100.116:8080/wardrobe/getClothing

请求参数：

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI2IiwiaWF0IjoxNjIwODM4MjExLCJzdWIiOi...
..	...

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

JSON

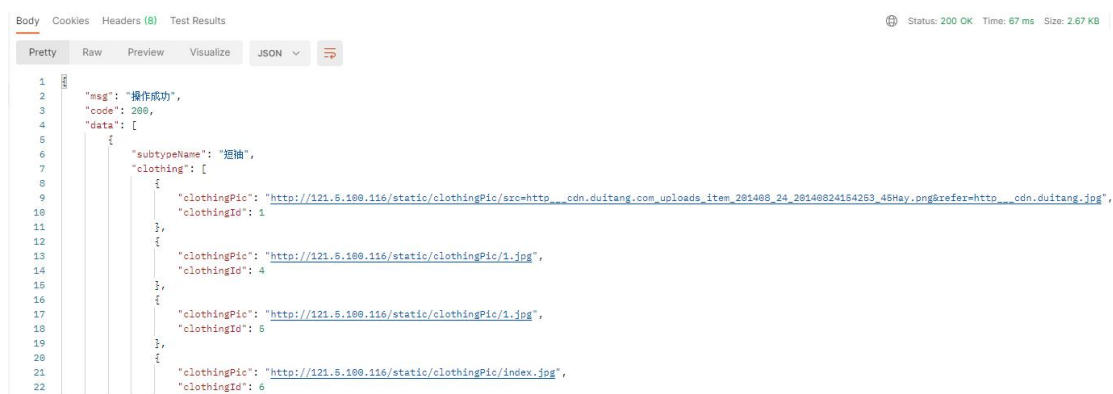
1

2

3

```
... "typeId":1
```

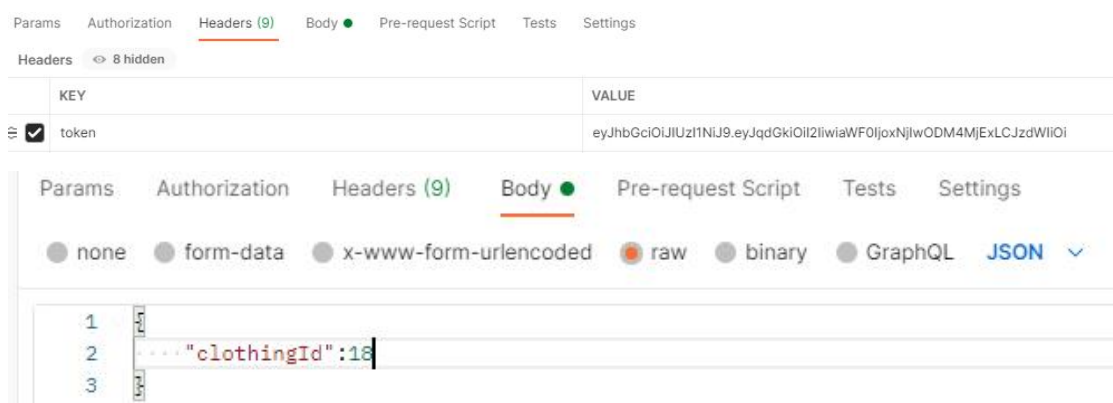
返回参数：



2.3 删除衣物

请求方法和路径: Post/121.5.100.116:8080/wardrobe/deleteClothing

请求参数:



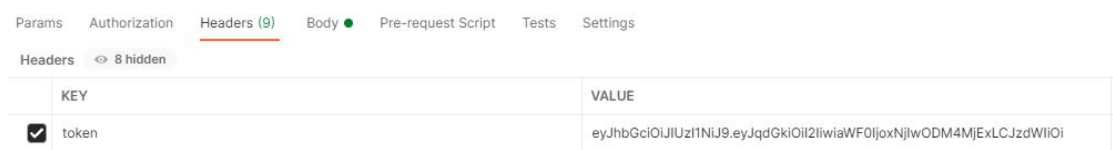
返回参数:



2.4 修改衣物类别

请求方法和路径: Post/121.5.100.116:8080/wardrobe/modifyClothing

请求参数:





返回参数:



2.5 获取类别静态表

请求方法和路径: Post/121.5.100.116:8080/wardrobe/modifyClothing

请求参数:

Headers 7 hidden	
KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI2IiwiaWF0IjoxNjIwODM4MjExLCJzdzWllOi

返回参数:

Body Cookies Headers (8) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1  {
2    "msg": "操作成功",
3    "code": 200,
4    "data": [
5      {
6        "subtype": [
7          {
8            "subtypeName": "短袖",
9            "subtypeId": 1
10         },
11         {
12           "subtypeName": "薄衫",
13           "subtypeId": 4
14         },
15         {
16           "subtypeName": "薄型T恤衫",
17           "subtypeId": 5
18         },
19         {
20           "subtypeName": "T恤衫",
21           "subtypeId": 7
22         }
23       ]
24     }
25   ]
26 }
```

3、“社区”模块（负责人:蔡瑞金）

3.1 获取用户所有博客

请求方法和路径：Post/121.5.100.116:8080/blog/getAll

请求参数：

Params	Authorization	Headers (9)	Body	Pre-request Script	Tests	Settings
Headers 8 hidden						
KEY		VALUE				
<input checked="" type="checkbox"/> token		eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiJzliwWFOljoXNjIwNzlyNTIiLCJzdzWlIiOi				
Key		Value				

返回参数：

Body 200 OK 282 ms 1.86 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "操作成功",
3   "code": 200,
4   "data": [
5     {
6       "blogPic": "/blogPic证件照.jpg",
7       "user_pic": "b",
8       "user_nickname": "蔡瑞金",
9       "blog_released_time": "2021-05-09 17:31:01",
10      "blogId": 1,
```

3.2 获取用户所有博客

请求方法和路径：Post/121.5.100.116:8080/blog/getCollection

请求参数：

Params	Authorization	Headers (9)	Body	Pre-request Script	Tests	Settings
Headers 8 hidden						
KEY		VALUE				
<input checked="" type="checkbox"/> token		eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiJzliwWFOljoXNjIwNzlyNTIiLCJzdzWlIiOi				

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   ... "userId": 3
3 }
4 }
```

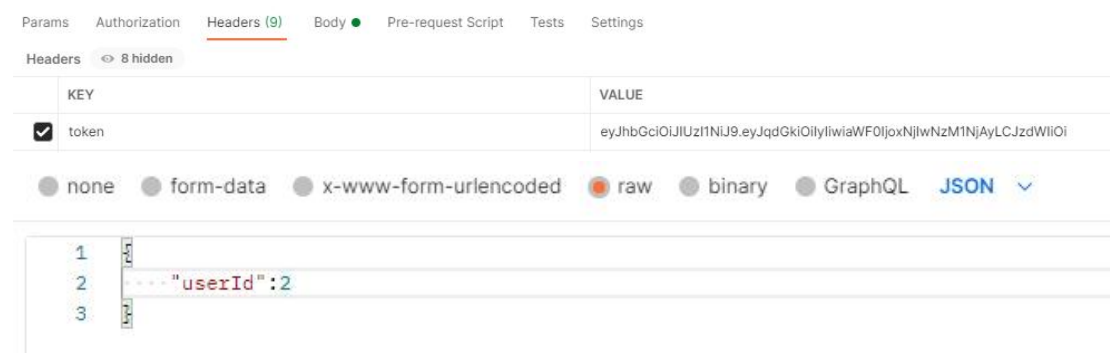
返回参数：



3.3 获取用户所有博客

请求方法和路径: Post/121.5.100.116:8080/user/getBlog

请求参数:



返回参数:



3.4 获取博客列表简略信息

请求方法和路径: Post/121.5.100.116:8080/blog/getIntro

请求参数:

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
token	eyJhbGciOiJIUzI1NiU9.eyJqdGkiOiJlIiwiaWF0IjoxNjwNzM1NjAyLCJzdWIIOi

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   ... "blogIdArray": [1,3,4]
3 }

```

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "msg": "操作成功",
3   "code": 200,
4   "data": [
5     {
6       "blogPic": "/blogPic证件照.jpg",
7       "favorite": 0,
8       "blogId": 1,
9       "userId": 3,
10      "blogTitle": "博客1,"
11    },
12    {
13      "blogPic": "C:\\Users\\Sakura\\Desktop\\blogPicu=309300470,3852732523&fm=224&gp=0.jpg",
14      "favorite": 0,
15      "blogId": 3,
16      "userId": 3,
17      "blogTitle": "jlsahk"
18    },
19    {
20      "blogPic": "C:\\Users\\Sakura\\Desktop\\blogPicu=309300470,3852732523&fm=224&gp=0.jpg",
21      "favorite": 0,
22      "blogId": 4,
23      "userId": 3,
24      "blogTitle": "jlsahk"
25    }
26  ]
27 }

```

3.5 获取博客详细信息

请求方法和路径: Post/121.5.100.116:8080/blog/getDetail

请求参数:

Headers 8 hidden

KEY	VALUE
token	eyJhbGciOiJIUzI1NiU9.eyJqdGkiOiJlIiwiaWF0IjoxNjwNzM1NjAyLCJzdWIIOi

none form-data x-www-form-urlencoded raw binary GraphQL JSON


```

1 {
2   ... "blogId": 6
3 }

```

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON 

```
1  {
2    "msg": "操作成功",
3    "code": 200,
4    "data": [
5      {
6        "blogPic": "/blogPic证件照.jpg",
7        "user_pic": "b",
8        "user_nickname": "蔡瑞金",
9        "blog_released_time": "2021-05-09 17:31:01",
10       "favorite": 0,
11       "blogId": 1,
12       "userId": 3,
13       "user_state": 2,
14       "blogTitle": "博客1,"
15     },
16     {
17       "blogPic": "C:\\Users\\Sakura\\Desktop\\blogPicu=309300470,3852732523&fm=224&gp=0.jpg",
18       "user_pic": "b",
19       "user_nickname": "蔡瑞金",
20       "blog_released_time": "2021-05-09 17:46:40",
21       "favorite": 0,
22       "blogId": 3,
23       "userId": 3,
24       "user_state": 2,
25       "blogTitle": "jlsahk"
26     },
27   ]
28 }
```

3.7 收藏/取消收藏博客

请求方法和路径: Post/121.5.100.116:8080/blog/collect

请求参数:

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIyIiwiaWF0IjoxNjM1NjAyLCJzdzWlI0i

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2  ... "blogId": 8
3
```

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1
2  "msg": "收藏成功",
3  "code": 200,
4  "data": ""
5
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1
2  "msg": "删除成功",
3  "code": 200,
4  "data": ""
5
```

3.8 搜索博客

请求方法和路径: Post/121.5.100.116:8080/blog/search

请求参数:

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIyIiwiaWF0IjoxNjM1NjAyLCJzdzWlI0i

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2  ... "keyword": "杰"
3
```

返回参数:

```
Body Cookies Headers (8) Test Results
Pretty Raw Preview Visualize JSON
1
2 "msg": "操作成功",
3 "code": 200,
4 "data": [
5   {
6     "blogPic": "null/outfits/blogPic/86273465_p0_master1200.jpg",
7     "user_pic": "sss",
8     "user_nickname": "金",
9     "blog_released_time": "2021-05-09 18:22:06",
10    "favorite": 0,
11    "userId": 2,
12    "blogId": 6,
13    "user_state": 1,
14    "blogTitle": "博客2"
15  }
16 ]
17
```

3.9 发布博客

请求方法和路径：Post/121.5.100.116:8080/blog/post

请求参数：

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiJpYXN0IiwiaWF0IjoxNjM1NjAyLCJzdzIWI0I

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> blogArticle	测试
<input checked="" type="checkbox"/> blogTitle	测试标题
<input checked="" type="checkbox"/> blogPic	aa.jpg ×

返回参数：

```
Body Cookies Headers (8) Test Results
Pretty Raw Preview Visualize JSON
1
2 "msg": "操作成功",
3 "code": 200,
4 "data": null
5
```

3.10 删除博客

请求方法和路径：Post/121.5.100.116:8080/blog/delete

请求参数：

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIyIiwiaWF0IjoxNjIwMzY1NjAyLCJzdzWiiOi

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

1

2

3

...

"blogId":11

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON ▼

1

2

3

4

5

...

"msg": "删除成功",

"code": 200,

"data": ""

4、搭配模块（负责人：221801210 林子鹏）

4.1 添加搭配

请求方法和路径: Post/121.5.100.116:8080/match/addMatch

请求参数:



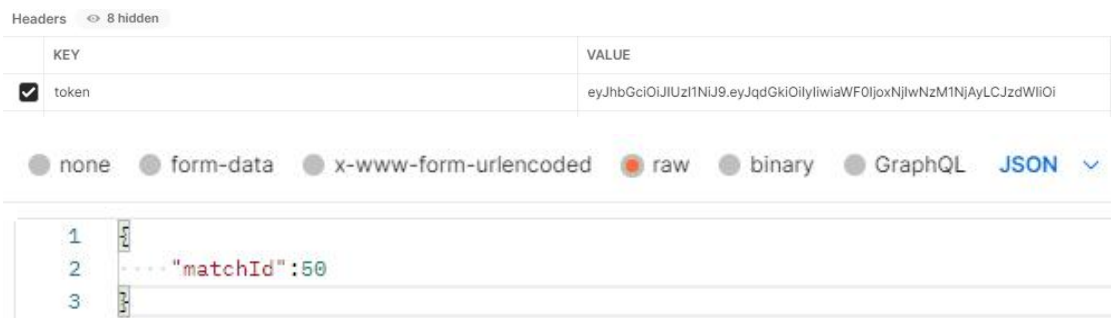
返回参数:



4.2 删除搭配

请求方法和路径: Post/121.5.100.116:8080/match/deleteMatch

请求参数:



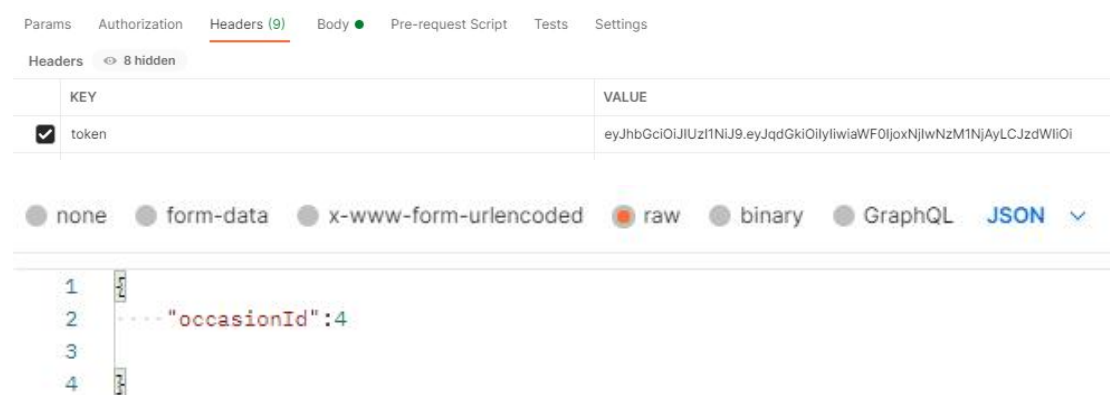
返回参数:



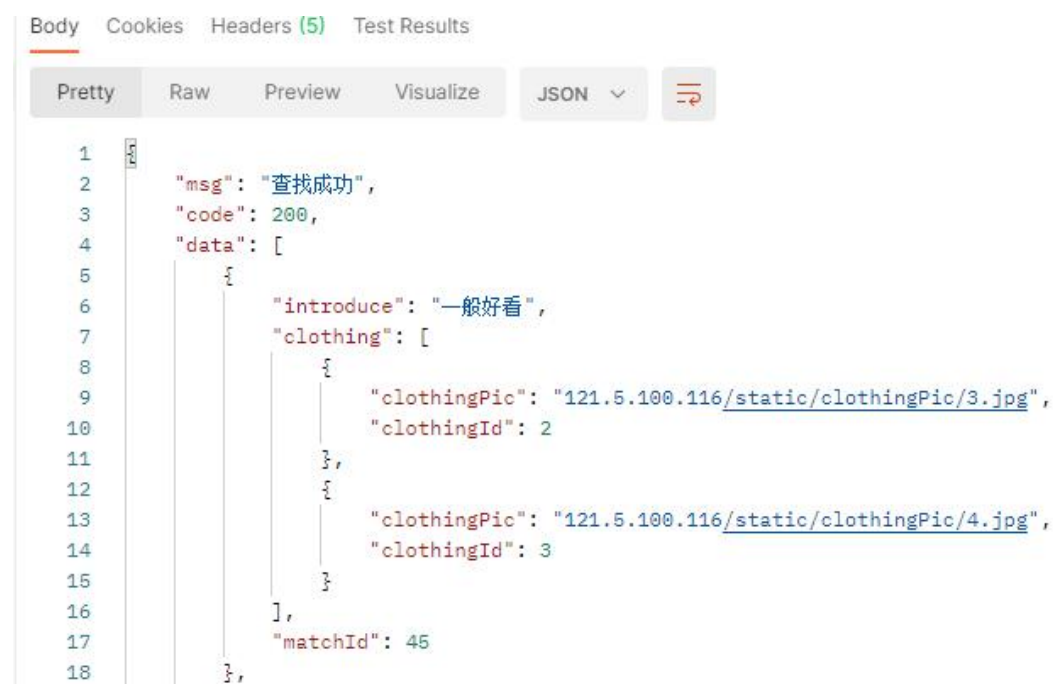
4.3 获取某场合所有搭配

请求方法和路径: Post/121.5.100.116:8080/match/listMatch

请求参数:



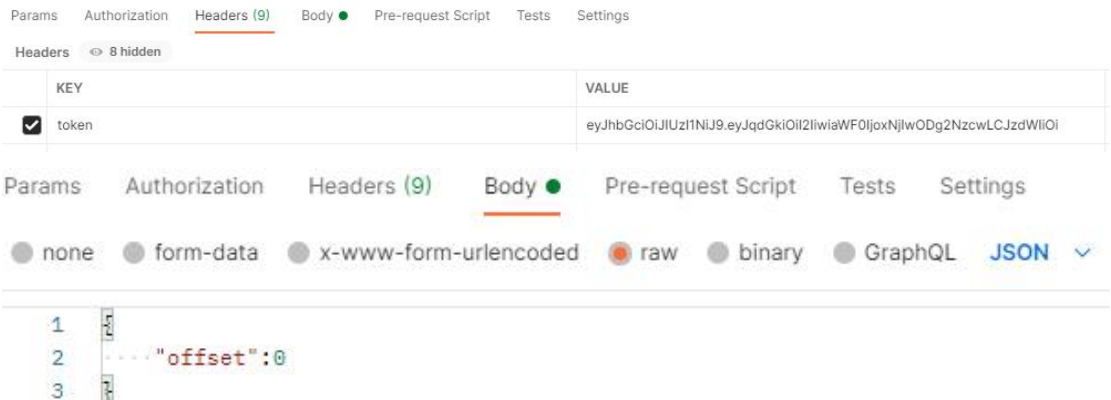
返回参数:



4.4 推荐搭配接口

请求方法和路径：Post/121.5.100.116:8080/match/recommand

请求参数：



(注:offset 为温度偏差值)

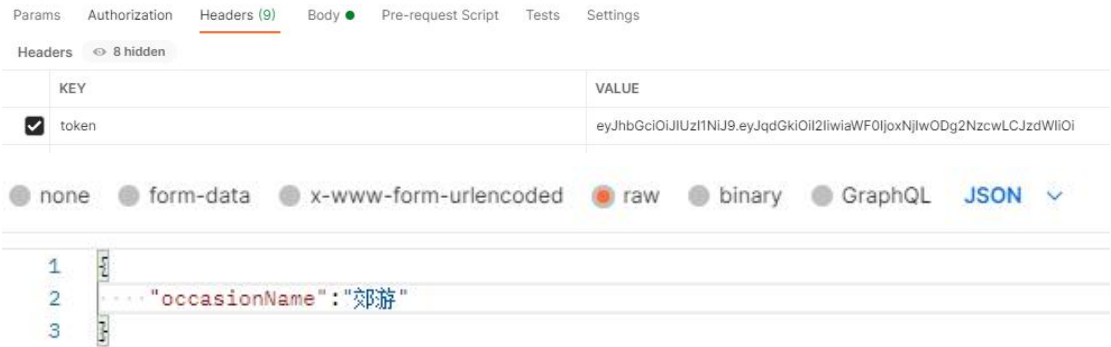
返回参数：



4.5 添加场合

请求方法和路径：Post/121.5.100.116:8080/match/addOccasion

请求参数：



返回参数：



4.6 删除场合:

请求方法和路径: Post/121.5.100.116:8080/match/deleteOccasion

请求参数:

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI2IiwiaWF0IjoxNjwODg2NzcwLCJzdWiiOi

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▾

1 2 3

... "occasionId":26

返回参数:

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON ▾

1 2 3 4 5

"msg": "删除成功",
"code": 200,
"data": ""

4.7 展示所有场合衣物:

请求方法和路径: Post/121.5.100.116:8080/match/listOccasion

请求参数:

Headers 7 hidden

KEY	VALUE
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI2IiwiaWF0IjoxNjwODg2NzcwLCJzdWiiOi

返回参数:

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1  {
2    "msg": "操作成功",
3    "code": 200,
4    "data": [
5      {
6        "occasionId": 4,
7        "occasionName": "打架"
8      },
9      {
10       "occasionId": 17,
11       "occasionName": "上课"
12     },
13     {
14       "occasionId": 18,
15       "occasionName": "上班"
16     },
17     {
18       "occasionId": 19,
19       "occasionName": "出去玩"
20     },
21   ]
22 }
```


Headers 8 hidden

返回参数:

Body

Cookies

Headers (8)

Test Results

Status: 200 OK

Time: 142 ms

Size: 468 B

Save Res

Pretty

Raw

Preview

Visualize

JSON

```
1 {  
2   "msg": "注册成功",  
3   "code": 200,  
4   "data": "eyJhbGciOiJIUzI1NiIsInR5cGE6ImF1dCI6ImVudG8zbnZyLjC3ZndwIGl0IiwiaXNTAOTQvODMwMCIsImVzcwyI6In3IiwiaWppbiI6ImV4c4iMTTYTE8NjkiLCJmcm90PnQHQSDGRdenH2w6ViLBGQPUeHqf2snH2w6VjYyo"  
5 }
```


二、单元测试

1.获取天气相关信息的工具类的测试（负责人：邱梓洛）

此接口返回数据为天气相关的数据，测试工作为手动测试，以下为测试的输入输出：

输入：无

输出：

• 天气推荐用例

城市	最高温	最低温	湿度	风速	月份	纬度	等级
福州	34	25	71	8.4	5	26.25	3
漳州	35	25	58	23.4	5	24.15	2
北京	26	15	55	23.4	5	39.93	1
成都	27	19	80	8.4	5	30.15	1
上海	26	20	85	3.0	5	30.67	1
泉州	32	25	72	23.4	5	24.90	2
莆田	32	25	67	23.4	5	22.51	2
厦门	33	24	74	23.4	5	24.45	2
三明	34	24	78	23.4	5	25.67	2
宁德	31	25	75	8.4	5	26.65	2
南平	33	24	75	23.4	5	26.63	2
杭州	27	21	86	23.4	5	29.56	1
深圳	32	26	74	8.4	5	22.53	2
石家庄	32	26	74	8.4	5	37.86	3
沈阳	21	13	63	23.4	5	42.41	0
南京	26	20	78	15.3	5	31.14	1

2. Md5 加密工具类的测试（221801239 林龙星）

Md5 加密生成的结果可以唯一确定，因此利用 Junit5 对此工具类进行测试。

测试代码：

```
✓ 测试结果 1 s 749 ms C:\Java\JDK\bin\java.exe ...
  ✓ TokenUtilTest 1 s 749 ms -----解析token-----
    ✓ parseJWT() 1 s 749 ms ID: 2
    Subject: 15260011385
    Issuer: ruijin
    IssuerAt: Tue May 11 20:20:02 CST 2021
    Expiration: Fri May 14 20:20:02 CST 2021
    currentTime: Fri May 14 12:28:37 CST 2021
    28284422 seconds is 0 days 7 hours 51 minutes and 2 seconds
    token的有效期小与48小时, 请更新token!

    进程已结束, 退出代码为 0
```

三、性能测试

1.社区模块

1.1 测试人员：221801239 林龙星

1.2 测试过程：首先为每个 Dao 层设置 100 个线程，每个线程运行十次。运行测试类查看运行所需时间。然后为每个 Dao 层设置 5000 个线程，每个线程运行 100 次，再查看运行时间，对比少并发和高并发的情况下运行时间的差异；由此分析出项目不足之处。

1.3 测试代码：

线程数:100 每个线程执行次数:10

```
@RunWith(SpringRunner.class)
@SpringBootTest
class BlogDaoTest {

    @Autowired
    public BlogDao blogDao;

    public void contextLoads(){

    }

    @Rule
    public ContiPerfRule contiPerfRule =new ContiPerfRule();

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void blogIdFindBlog() {
        List<Blog> blogList = blogDao.blogIdFindBlog(1);
        System.out.println(blogList.get(0));
    }

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void userFindSubscribe() {
        List<SubscribeUser> subscribeUsers = blogDao.userFindSubscribe( user_id: 2);
        System.out.println(subscribeUsers.get(0));
    }

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void userFindBlog() {
        List<Blog> blogList = blogDao.userFindBlog( user_id: 2);
        System.out.println(blogList.get(0));
    }

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void userBlogFind() {
        int totalNum = blogDao.userBlogFind( user_id: 2, blog_id: 2);
        System.out.println(totalNum);
    }

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void articleSearch() {
        List<Blog> blogList = blogDao.articleSearch( keyword: "测试");
    }

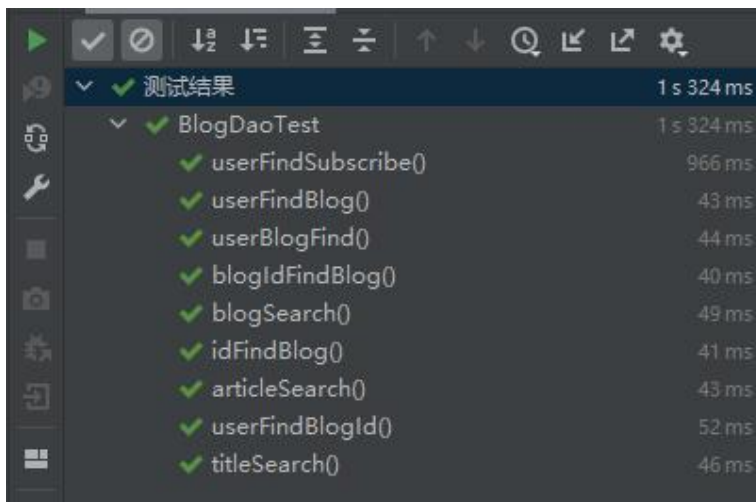
    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void titleSearch() {
        List<Blog> blogList = blogDao.titleSearch( keyword: "测试");
    }

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void blogSearch() {
        List<Blog> blogList = blogDao.blogSearch();
        System.out.println(blogList.get(0));
    }

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void userFindBlogId() {
        List<CollectedBlog> collectedBlogs = blogDao.userFindBlogId( user_id: 1);
    }

    @Test
    @PerfTest(invocations = 10,threads = 100)//设置100个线程，每个线程运行十次
    void idFindBlog() {
        List<Blog> blogList = blogDao.IdFindBlog(2);
    }
}
```

执行结果:



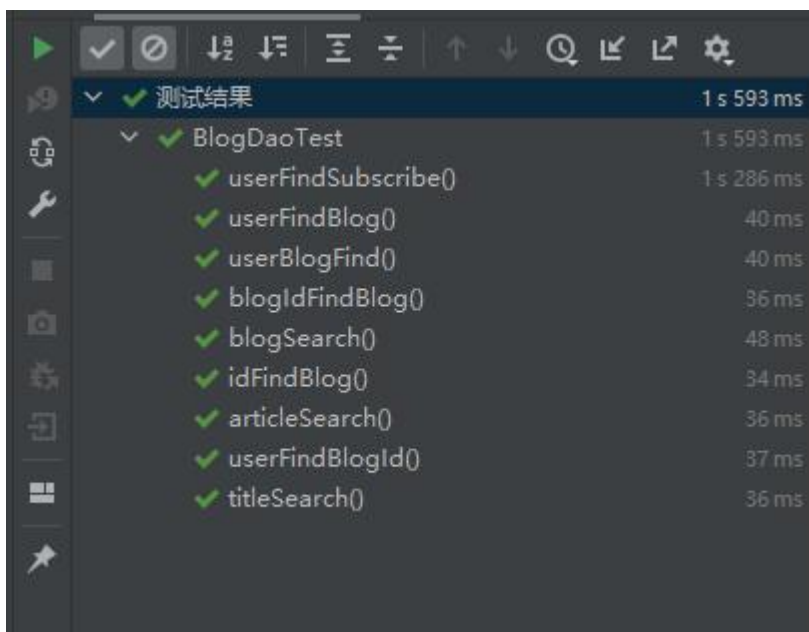
测试结果	1 s 324 ms
BlogDaoTest	1 s 324 ms
userFindSubscribe()	966 ms
userFindBlog()	43 ms
userBlogFind()	44 ms
blogIdFindBlog()	40 ms
blogSearch()	49 ms
idFindBlog()	41 ms
articleSearch()	43 ms
userFindBlogId()	52 ms
titleSearch()	46 ms

线程数:5000 每个线程执行次数:100

以下为部分代码，与上面代码差别就在于线程和线程执行次数的区别

```
@Test
@PerfTest(invocations = 100, threads = 5000) // 设置1000个线程，每个线程运行100次
void userFindSubscribe() {
    List<SubscribeUser> subscribeUsers = blogDao.userFindSubscribe( user_id: 2);
    System.out.println(subscribeUsers.get(0));
}
```

执行结果：



测试结果	1 s 593 ms
BlogDaoTest	1 s 593 ms
userFindSubscribe()	1 s 286 ms
userFindBlog()	40 ms
userBlogFind()	40 ms
blogIdFindBlog()	36 ms
blogSearch()	48 ms
idFindBlog()	34 ms
articleSearch()	36 ms
userFindBlogId()	37 ms
titleSearch()	36 ms

1.4 结果分析：从以上测试数据上看，社区部分的 Dao 主要耗时的部分是获取用户关注的用户的接口。其他部分接口耗时相对较少，而这有可能是因为数据库中的数据量不够导致，后期我们还会持续对项目进行相关测试，以找出真正问题所在。

2.我的模块

2.1 测试人员：221801239 林龙星

2.2 测试过程：首先为每个 Dao 层设置 200 个线程，每个线程运行 20 次。运行测试类查看运行所需时间。然后为每个 Dao 层设置 5000 个线程，每个线程运行 200 次，对比两种情况下运行时间的差异；由此分析出那个接口耗时最多。

2.3 测试代码：

线程数:200 每个线程执行次数:20

```
package com.example.backendframework.Dao.meDao;

import ...

@RunWith(SpringRunner.class)
@SpringBootTest
class SubscribeDaoTest {

    @Autowired
    public SubscribeDao subscribeDao;

    public void contextLoads(){

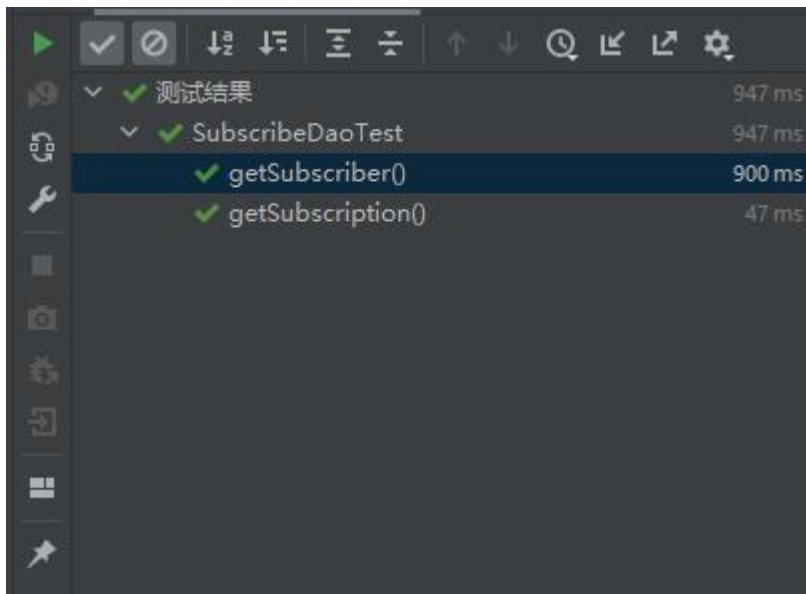
    }

    @Rule
    public ContiPerfRule contiPerfRule =new ContiPerfRule();

    @Test
    @PerfTest(invocations = 20,threads = 200)//设置200个线程，每个线程运行20次
    void getSubscription() {
        List<User> userList = subscribeDao.getSubscription( user_id: 2);
        System.out.println(userList.get(0));
    }

    @Test
    @PerfTest(invocations = 20,threads = 200)//设置200个线程，每个线程运行20次
    void getSubscriber() {
        List<User> userList = subscribeDao.getSubscriber( user_id: 3);
        System.out.println(userList.get(0));
    }
}
```

执行结果:



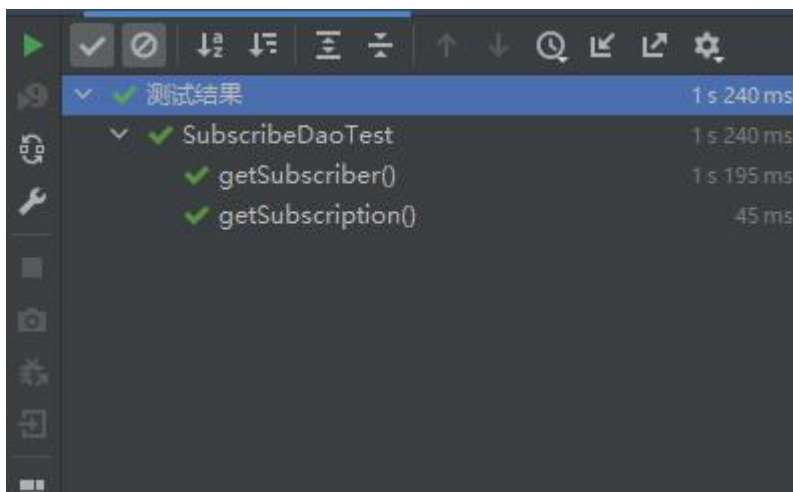
线程数:5000 每个线程执行次数:200

以下为部分代码，与上面代码差别就在于线程和线程执行次数的区别

```
@Rule
public ContiPerfRule contiPerfRule = new ContiPerfRule();

@Test
@PerfTest(invocations = 200, threads = 5000) // 设置5000个线程，每个线程运行200次
void getSubscription() {
    List<User> userList = subscribeDao.getSubscription( user_id: 2);
    System.out.println(userList.get(0));
}
```

执行结果：



2.4 结果分析：从以上测试数据上看，我的部分的 Dao 主要耗时的部分也是获取用户关注的用户的接口。其他部分接口耗时相对较少。可以考虑优化获取用户关注的用户的接口。

3.衣柜模块

3.1 测试人员：221801239 林龙星

3.2 测试过程：首先为每个 Dao 层设置 300 个线程，每个线程运行 30 次。运行测试类查看运行所需时间。然后为每个 Dao 层设置 5000 个线程，每个线程运行 300 次，对比两种情况下运行时间的差异；由此分析出那个接口耗时最多。

3.3 测试代码：

线程数:300 每个线程执行次数:30

```
@SpringBootTest
class WardrobeDaoTest {
    @Autowired
    public WardrobeDao wardrobeDao;

    public void contextLoads(){
    }

    @Rule
    public ContiPerfRule contiPerfRule =new ContiPerfRule();

    @Test
    @PerfTest(invocations = 30,threads = 300)//设置300个线程，每个线程运行30次
    void getSubtype() {
        List<Subtype> subtypes = wardrobeDao.getSubtype( typeid: 2);
    }

    @Test
    @PerfTest(invocations = 30,threads = 300)//设置300个线程，每个线程运行30次
    void getClothing() {
        List<Clothing> clothing = wardrobeDao.getClothing( userid: 2, subtypeId: 2);
    }

    @Test
    @PerfTest(invocations = 30,threads = 300)//设置300个线程，每个线程运行30次
    void getType() {
        List<Type> types = wardrobeDao.getType();
    }

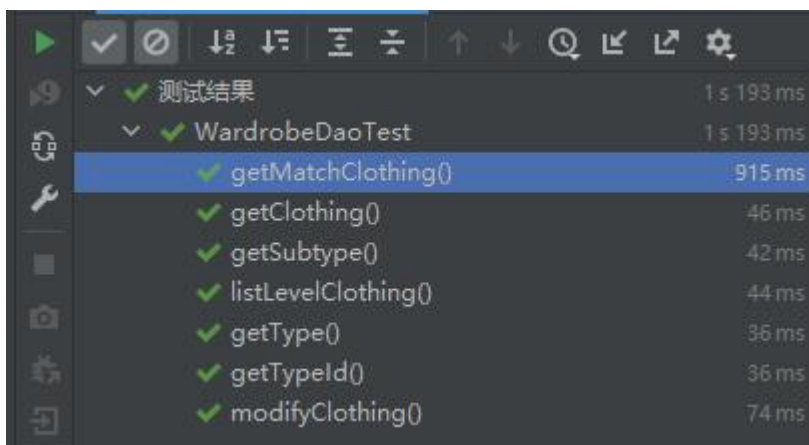
    @Test
    @PerfTest(invocations = 30,threads = 300)//设置300个线程，每个线程运行30次
    void modifyClothing() {
        wardrobeDao.modifyClothing( clothingId: 2, subtypeId: 2);
    }

    @Test
    @PerfTest(invocations = 30,threads = 300)//设置300个线程，每个线程运行30次
    void getMatchClothing() {
        List<MatchClothing> matchClothings = wardrobeDao.getMatchClothing( clothingId: 2);
    }

    @Test
    @PerfTest(invocations = 30,threads = 300)//设置300个线程，每个线程运行30次
    void listLevelClothing() {
        List<Clothing> listLevelClothing = wardrobeDao.listLevelClothing( userid: 2, level: 3);
    }

    @Test
    @PerfTest(invocations = 30,threads = 300)//设置300个线程，每个线程运行30次
    void getTypeId() {
        int num = wardrobeDao.getTypeId( subtype_id: 2);
    }
}
```

执行结果:



测试项	耗时
测试结果	1 s 193 ms
WardrobeDaoTest	1 s 193 ms
getMatchClothing()	915 ms
getClothing()	46 ms
getSubtype()	42 ms
listLevelClothing()	44 ms
getType()	36 ms
getTypeId()	36 ms
modifyClothing()	74 ms

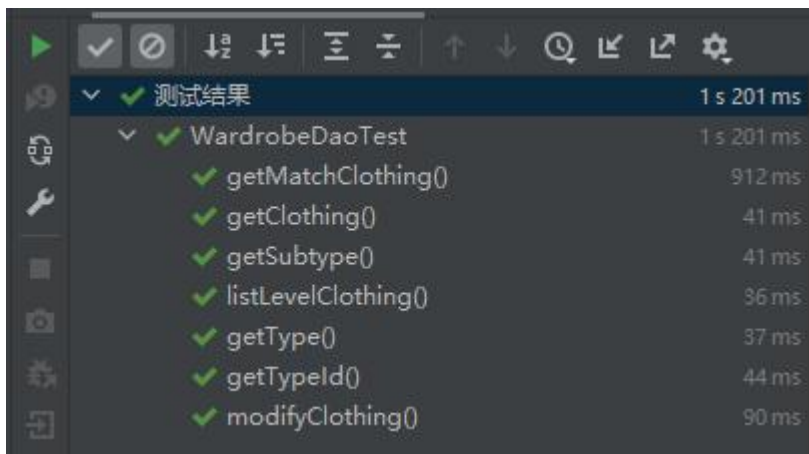
线程数:5000 每个线程执行次数:300

以下为部分代码，与上面代码差别就在于线程和线程执行次数的区别

```
@Rule
public ContiPerfRule contiPerfRule = new ContiPerfRule();

@Test
@PerfTest(invocations = 300, threads = 5000) // 设置5000个线程，每个线程运行300次
void getSubtype() {
    List<Subtype> subtypes = wardrobeDao.getSubtype( typeId: 2);
}
```

执行结果：



测试项	耗时
测试结果	1 s 201 ms
WardrobeDaoTest	1 s 201 ms
getMatchClothing()	912 ms
getClothing()	41 ms
getSubtype()	41 ms
listLevelClothing()	36 ms
getType()	37 ms
getTypeId()	44 ms
modifyClothing()	90 ms

3.4 结果分析：从以上测试数据上看，衣柜部分的 Dao 主要耗时的部分是获取搭配衣物的接口。其他部分接口耗时相对较少。在高并发情况下运行时间并没有明显增加，可能是数据库内衣物数据不足，后续还会继续观察该部分的性能。

4.搭配模块

4.1 测试人员：221801239 林龙星

4.2 测试过程：首先为每个 Dao 层设置 400 个线程，每个线程运行 40 次。运行测试类查看运行所需时间。然后为每个 Dao 层设置 5000 个线程，每个线程运行 400 次，对比两种情况下运行时间的差异；由此分析出那个接口耗时最多。

4.3 测试代码：

线程数:400 每个线程执行次数:40

```
@RunWith(SpringRunner.class)
@SpringBootTest
class MatchDaoTest {
    @Autowired
    public MatchDao matchDao;

    @Autowired
    public MatchClothingDao matchClothingDao;

    @Autowired
    public OccasionDao occasionDao ;

    public void contextLoads(){

    }

    @Rule
    public ContiPerfRule contiPerfRule =new ContiPerfRule();

    @Test
    @PerfTest(invocations = 40,threads = 400)//设置400个线程，每个线程运行40次
    void existOccasion() {
        int num = matchDao.existOccasion( occasion_id: 2);
    }

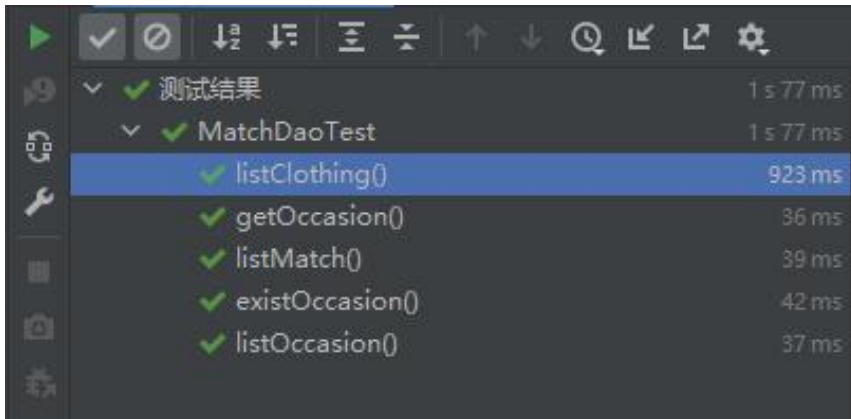
    @Test
    @PerfTest(invocations = 40,threads = 400)//设置400个线程，每个线程运行40次
    void listMatch() {
        List<Match> matches = matchDao.listMatch( occasion_id: 2);
    }

    @Test
    @PerfTest(invocations = 40,threads = 400)//设置400个线程，每个线程运行40次
    void listClothing() {
        List<Clothing> clothing = matchClothingDao.listClothing( match_id: 2);
    }

    @Test
    @PerfTest(invocations = 40,threads = 400)//设置400个线程，每个线程运行40次
    void listOccasion(){
        List<Occasion> occasions = occasionDao.listOccasion( user_id: 2);
    }

    @Test
    @PerfTest(invocations = 40,threads = 400)//设置400个线程，每个线程运行40次
    void getOccasion(){
        Occasion occasion = occasionDao.getOccasion( occasion_id: 2);
    }
}
```

执行结果:



The screenshot shows the test results for the MatchDaoTest class. The test suite passed, and the individual methods and their execution times are listed.

Test Method	Execution Time
listClothing()	923 ms
getOccasion()	36 ms
listMatch()	39 ms
existOccasion()	42 ms
listOccasion()	37 ms

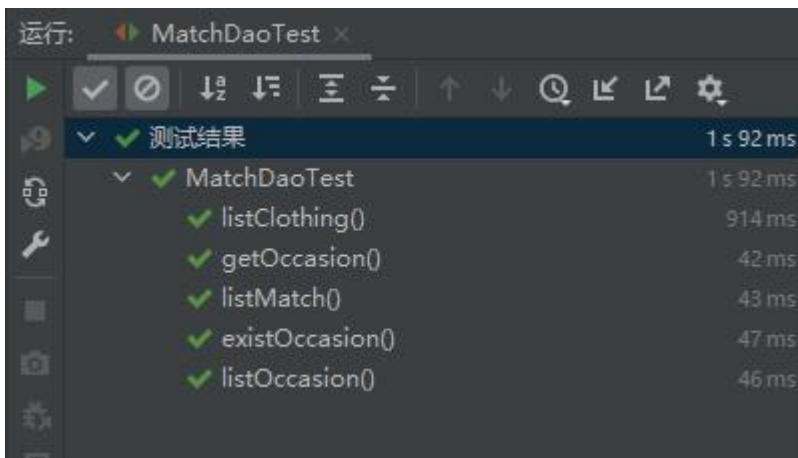
线程数:5000 每个线程执行次数:400

以下为部分代码, 与上面代码差别就在于线程和线程执行次数的区别

```
@Rule
public ContiPerfRule contiPerfRule = new ContiPerfRule();

@Test
@PerfTest(invocations = 500, threads = 5000) // 设置5000个线程, 每个线程运行500次
void existOccasion() {
    int num = matchDao.existOccasion(occasion_id: 2);
}
```

执行结果:



The screenshot shows the test results for the MatchDaoTest class with 5000 threads. The test suite passed, and the individual methods and their execution times are listed.

Test Method	Execution Time
listClothing()	914 ms
getOccasion()	42 ms
listMatch()	43 ms
existOccasion()	47 ms
listOccasion()	46 ms

4.4 结果分析: 从以上测试数据上看, 搭配部分的 Dao 主要耗时的部分是获取衣物列表的接口。其他部分接口耗时相对较少。在高并发情况下运行时间并没有明显增加, 考虑到后期数据库衣物相关的数据会较多, 后期可能查询数据会消耗大量时间, 所有后面我们可能会想办法优化。