**springboot项目整合resttemplate（调用第三方接口）**



[乘虚](https://blog.csdn.net/hx1024877209)于 2023-04-24 14:02:38 发布

阅读量2.2k 收藏 15

点赞数

文章标签： [spring boot](https://so.csdn.net/so/search/s.do?q=spring+boot&t=all&o=vip&s=&l=&f=&viparticle=&from_tracking_code=tag_word&from_code=app_blog_art) [java](https://so.csdn.net/so/search/s.do?q=java&t=all&o=vip&s=&l=&f=&viparticle=&from_tracking_code=tag_word&from_code=app_blog_art) [后端](https://so.csdn.net/so/search/s.do?q=%E5%90%8E%E7%AB%AF&t=all&o=vip&s=&l=&f=&viparticle=&from_tracking_code=tag_word&from_code=app_blog_art)

版权

**1.首先添加maven依赖（应该已经有了）**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

**2.增加一个 RestTemplateConfig 类**

1. @Configuration
2. public class RestTemplateConfig {
3. @Bean
4. public RestTemplate restTemplate(ClientHttpRequestFactory factory){
5. return new RestTemplate(factory);
6. }
8. @Bean
9. public ClientHttpRequestFactory simpleClientHttpRequestFactory(){
10. SimpleClientHttpRequestFactory factory = new SimpleClientHttpRequestFactory();
11. factory.setConnectTimeout(15000);
12. factory.setReadTimeout(5000);
13. return factory;
14. }
15. }

**3.增加一个  RestTemplateUtil 类（工具类）**

1. package com.example.test001.util;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.http.HttpEntity;
5. import org.springframework.http.HttpHeaders;
6. import org.springframework.http.HttpMethod;
7. import org.springframework.http.ResponseEntity;
8. import org.springframework.stereotype.Component;
9. import org.springframework.web.client.RestTemplate;
11. import java.util.Map;
13. */\*\**
14. *\* @author hx*
15. *\* @create 2023/4/24 10:16*
16. *\* description:*
17. *\*/*
18. @Component
19. public class RestTemplateUtil {
20. @Autowired
21. private RestTemplate restTemplate;


25. *// ----------------------------------GET-------------------------------------------------------*
27. */\*\**
28. *\* GET请求调用方式*
29. *\**
30. *\* @param url 请求URL*
31. *\* @param responseType 返回对象类型*
32. *\* @return ResponseEntity 响应对象封装类*
33. *\*/*
34. public <T> ResponseEntity<T> get(String url, Class<T> responseType) {
35. return restTemplate.getForEntity(url, responseType);
36. }
38. */\*\**
39. *\* GET请求调用方式*
40. *\**
41. *\* @param url 请求URL*
42. *\* @param responseType 返回对象类型*
43. *\* @param uriVariables URL中的变量，按顺序依次对应*
44. *\* @return ResponseEntity 响应对象封装类*
45. *\*/*
46. public <T> ResponseEntity<T> get(String url, Class<T> responseType, Object... uriVariables) {
47. return restTemplate.getForEntity(url, responseType, uriVariables);
48. }
50. */\*\**
51. *\* GET请求调用方式*
52. *\**
53. *\* @param url 请求URL*
54. *\* @param responseType 返回对象类型*
55. *\* @param uriVariables URL中的变量，与Map中的key对应*
56. *\* @return ResponseEntity 响应对象封装类*
57. *\*/*
58. public <T> ResponseEntity<T> get(String url, Class<T> responseType, Map<String, ?> uriVariables) {
59. return restTemplate.getForEntity(url, responseType, uriVariables);
60. }
62. */\*\**
63. *\* 带请求头的GET请求调用方式*
64. *\**
65. *\* @param url 请求URL*
66. *\* @param headers 请求头参数*
67. *\* @param responseType 返回对象类型*
68. *\* @param uriVariables URL中的变量，按顺序依次对应*
69. *\* @return ResponseEntity 响应对象封装类*
70. *\*/*
71. public <T> ResponseEntity<T> get(String url, Map<String, String> headers, Class<T> responseType, Object... uriVariables) {
72. HttpHeaders httpHeaders = new HttpHeaders();
73. httpHeaders.setAll(headers);
74. return get(url, httpHeaders, responseType, uriVariables);
75. }
77. */\*\**
78. *\* 带请求头的GET请求调用方式*
79. *\**
80. *\* @param url 请求URL*
81. *\* @param headers 请求头参数*
82. *\* @param responseType 返回对象类型*
83. *\* @param uriVariables URL中的变量，按顺序依次对应*
84. *\* @return ResponseEntity 响应对象封装类*
85. *\*/*
86. public <T> ResponseEntity<T> get(String url, HttpHeaders headers, Class<T> responseType, Object... uriVariables) {
87. HttpEntity<?> requestEntity = new HttpEntity<>(headers);
88. return exchange(url, HttpMethod.GET, requestEntity, responseType, uriVariables);
89. }
91. */\*\**
92. *\* 带请求头的GET请求调用方式*
93. *\**
94. *\* @param url 请求URL*
95. *\* @param headers 请求头参数*
96. *\* @param responseType 返回对象类型*
97. *\* @param uriVariables URL中的变量，与Map中的key对应*
98. *\* @return ResponseEntity 响应对象封装类*
99. *\*/*
100. public <T> ResponseEntity<T> get(String url, Map<String, String> headers, Class<T> responseType, Map<String, ?> uriVariables) {
101. HttpHeaders httpHeaders = new HttpHeaders();
102. httpHeaders.setAll(headers);
103. return get(url, httpHeaders, responseType, uriVariables);
104. }
106. */\*\**
107. *\* 带请求头的GET请求调用方式*
108. *\**
109. *\* @param url 请求URL*
110. *\* @param headers 请求头参数*
111. *\* @param responseType 返回对象类型*
112. *\* @param uriVariables URL中的变量，与Map中的key对应*
113. *\* @return ResponseEntity 响应对象封装类*
114. *\*/*
115. public <T> ResponseEntity<T> get(String url, HttpHeaders headers, Class<T> responseType, Map<String, ?> uriVariables) {
116. HttpEntity<?> requestEntity = new HttpEntity<>(headers);
117. return exchange(url, HttpMethod.GET, requestEntity, responseType, uriVariables);
118. }
120. *// ----------------------------------POST-------------------------------------------------------*
122. */\*\**
123. *\* POST请求调用方式*
124. *\**
125. *\* @param url 请求URL*
126. *\* @param responseType 返回对象类型*
127. *\* @return*
128. *\*/*
129. public <T> ResponseEntity<T> post(String url, Class<T> responseType) {
130. return restTemplate.postForEntity(url, HttpEntity.EMPTY, responseType);
131. }
133. */\*\**
134. *\* POST请求调用方式*
135. *\**
136. *\* @param url 请求URL*
137. *\* @param requestBody 请求参数体*
138. *\* @param responseType 返回对象类型*
139. *\* @return ResponseEntity 响应对象封装类*
140. *\*/*
141. public <T> ResponseEntity<T> post(String url, Object requestBody, Class<T> responseType) {
142. return restTemplate.postForEntity(url, requestBody, responseType);
143. }
145. */\*\**
146. *\* POST请求调用方式*
147. *\**
148. *\* @param url 请求URL*
149. *\* @param requestBody 请求参数体*
150. *\* @param responseType 返回对象类型*
151. *\* @param uriVariables URL中的变量，按顺序依次对应*
152. *\* @return ResponseEntity 响应对象封装类*
153. *\*/*
154. public <T> ResponseEntity<T> post(String url, Object requestBody, Class<T> responseType, Object... uriVariables) {
155. return restTemplate.postForEntity(url, requestBody, responseType, uriVariables);
156. }
158. */\*\**
159. *\* POST请求调用方式*
160. *\**
161. *\* @param url 请求URL*
162. *\* @param requestBody 请求参数体*
163. *\* @param responseType 返回对象类型*
164. *\* @param uriVariables URL中的变量，与Map中的key对应*
165. *\* @return ResponseEntity 响应对象封装类*
166. *\*/*
167. public <T> ResponseEntity<T> post(String url, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
168. return restTemplate.postForEntity(url, requestBody, responseType, uriVariables);
169. }
171. */\*\**
172. *\* 带请求头的POST请求调用方式*
173. *\**
174. *\* @param url 请求URL*
175. *\* @param headers 请求头参数*
176. *\* @param requestBody 请求参数体*
177. *\* @param responseType 返回对象类型*
178. *\* @param uriVariables URL中的变量，按顺序依次对应*
179. *\* @return ResponseEntity 响应对象封装类*
180. *\*/*
181. public <T> ResponseEntity<T> post(String url, Map<String, String> headers, Object requestBody, Class<T> responseType, Object... uriVariables) {
182. HttpHeaders httpHeaders = new HttpHeaders();
183. httpHeaders.setAll(headers);
184. return post(url, httpHeaders, requestBody, responseType, uriVariables);
185. }
187. */\*\**
188. *\* 带请求头的POST请求调用方式*
189. *\**
190. *\* @param url 请求URL*
191. *\* @param headers 请求头参数*
192. *\* @param requestBody 请求参数体*
193. *\* @param responseType 返回对象类型*
194. *\* @param uriVariables URL中的变量，按顺序依次对应*
195. *\* @return ResponseEntity 响应对象封装类*
196. *\*/*
197. public <T> ResponseEntity<T> post(String url, HttpHeaders headers, Object requestBody, Class<T> responseType, Object... uriVariables) {
198. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody, headers);
199. return post(url, requestEntity, responseType, uriVariables);
200. }
202. */\*\**
203. *\* 带请求头的POST请求调用方式*
204. *\**
205. *\* @param url 请求URL*
206. *\* @param headers 请求头参数*
207. *\* @param requestBody 请求参数体*
208. *\* @param responseType 返回对象类型*
209. *\* @param uriVariables URL中的变量，与Map中的key对应*
210. *\* @return ResponseEntity 响应对象封装类*
211. *\*/*
212. public <T> ResponseEntity<T> post(String url, Map<String, String> headers, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
213. HttpHeaders httpHeaders = new HttpHeaders();
214. httpHeaders.setAll(headers);
215. return post(url, httpHeaders, requestBody, responseType, uriVariables);
216. }
217. */\*\**
218. *\* 带请求头的POST请求调用方式*
219. *\**
220. *\* @param url 请求URL*
221. *\* @param headers 请求头参数*
222. *\* @param requestBody 请求参数体*
223. *\* @param responseType 返回对象类型*
224. *\* @param uriVariables URL中的变量，与Map中的key对应*
225. *\* @return ResponseEntity 响应对象封装类*
226. *\*/*
227. public <T> ResponseEntity<T> post(String url, HttpHeaders headers, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
228. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody, headers);
229. return post(url, requestEntity, responseType, uriVariables);
230. }
232. */\*\**
233. *\* 自定义请求头和请求体的POST请求调用方式*
234. *\**
235. *\* @param url 请求URL*
236. *\* @param requestEntity 请求头和请求体封装对象*
237. *\* @param responseType 返回对象类型*
238. *\* @param uriVariables URL中的变量，按顺序依次对应*
239. *\* @return ResponseEntity 响应对象封装类*
240. *\*/*
241. public <T> ResponseEntity<T> post(String url, HttpEntity<?> requestEntity, Class<T> responseType, Object... uriVariables) {
242. return restTemplate.exchange(url, HttpMethod.POST, requestEntity, responseType, uriVariables);
243. }
245. */\*\**
246. *\* 自定义请求头和请求体的POST请求调用方式*
247. *\**
248. *\* @param url 请求URL*
249. *\* @param requestEntity 请求头和请求体封装对象*
250. *\* @param responseType 返回对象类型*
251. *\* @param uriVariables URL中的变量，与Map中的key对应*
252. *\* @return ResponseEntity 响应对象封装类*
253. *\*/*
254. public <T> ResponseEntity<T> post(String url, HttpEntity<?> requestEntity, Class<T> responseType, Map<String, ?> uriVariables) {
255. return restTemplate.exchange(url, HttpMethod.POST, requestEntity, responseType, uriVariables);
256. }
258. *// ----------------------------------PUT-------------------------------------------------------*
260. */\*\**
261. *\* PUT请求调用方式*
262. *\**
263. *\* @param url 请求URL*
264. *\* @param responseType 返回对象类型*
265. *\* @param uriVariables URL中的变量，按顺序依次对应*
266. *\* @return ResponseEntity 响应对象封装类*
267. *\*/*
268. public <T> ResponseEntity<T> put(String url, Class<T> responseType, Object... uriVariables) {
269. return put(url, HttpEntity.EMPTY, responseType, uriVariables);
270. }
272. */\*\**
273. *\* PUT请求调用方式*
274. *\**
275. *\* @param url 请求URL*
276. *\* @param requestBody 请求参数体*
277. *\* @param responseType 返回对象类型*
278. *\* @param uriVariables URL中的变量，按顺序依次对应*
279. *\* @return ResponseEntity 响应对象封装类*
280. *\*/*
281. public <T> ResponseEntity<T> put(String url, Object requestBody, Class<T> responseType, Object... uriVariables) {
282. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody);
283. return put(url, requestEntity, responseType, uriVariables);
284. }
286. */\*\**
287. *\* PUT请求调用方式*
288. *\**
289. *\* @param url 请求URL*
290. *\* @param requestBody 请求参数体*
291. *\* @param responseType 返回对象类型*
292. *\* @param uriVariables URL中的变量，与Map中的key对应*
293. *\* @return ResponseEntity 响应对象封装类*
294. *\*/*
295. public <T> ResponseEntity<T> put(String url, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
296. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody);
297. return put(url, requestEntity, responseType, uriVariables);
298. }
300. */\*\**
301. *\* 带请求头的PUT请求调用方式*
302. *\**
303. *\* @param url 请求URL*
304. *\* @param headers 请求头参数*
305. *\* @param requestBody 请求参数体*
306. *\* @param responseType 返回对象类型*
307. *\* @param uriVariables URL中的变量，按顺序依次对应*
308. *\* @return ResponseEntity 响应对象封装类*
309. *\*/*
310. public <T> ResponseEntity<T> put(String url, Map<String, String> headers, Object requestBody, Class<T> responseType, Object... uriVariables) {
311. HttpHeaders httpHeaders = new HttpHeaders();
312. httpHeaders.setAll(headers);
313. return put(url, httpHeaders, requestBody, responseType, uriVariables);
314. }
316. */\*\**
317. *\* 带请求头的PUT请求调用方式*
318. *\**
319. *\* @param url 请求URL*
320. *\* @param headers 请求头参数*
321. *\* @param requestBody 请求参数体*
322. *\* @param responseType 返回对象类型*
323. *\* @param uriVariables URL中的变量，按顺序依次对应*
324. *\* @return ResponseEntity 响应对象封装类*
325. *\*/*
326. public <T> ResponseEntity<T> put(String url, HttpHeaders headers, Object requestBody, Class<T> responseType, Object... uriVariables) {
327. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody, headers);
328. return put(url, requestEntity, responseType, uriVariables);
329. }
331. */\*\**
332. *\* 带请求头的PUT请求调用方式*
333. *\**
334. *\* @param url 请求URL*
335. *\* @param headers 请求头参数*
336. *\* @param requestBody 请求参数体*
337. *\* @param responseType 返回对象类型*
338. *\* @param uriVariables URL中的变量，与Map中的key对应*
339. *\* @return ResponseEntity 响应对象封装类*
340. *\*/*
341. public <T> ResponseEntity<T> put(String url, Map<String, String> headers, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
342. HttpHeaders httpHeaders = new HttpHeaders();
343. httpHeaders.setAll(headers);
344. return put(url, httpHeaders, requestBody, responseType, uriVariables);
345. }
347. */\*\**
348. *\* 带请求头的PUT请求调用方式*
349. *\**
350. *\* @param url 请求URL*
351. *\* @param headers 请求头参数*
352. *\* @param requestBody 请求参数体*
353. *\* @param responseType 返回对象类型*
354. *\* @param uriVariables URL中的变量，与Map中的key对应*
355. *\* @return ResponseEntity 响应对象封装类*
356. *\*/*
357. public <T> ResponseEntity<T> put(String url, HttpHeaders headers, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
358. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody, headers);
359. return put(url, requestEntity, responseType, uriVariables);
360. }
362. */\*\**
363. *\* 自定义请求头和请求体的PUT请求调用方式*
364. *\**
365. *\* @param url 请求URL*
366. *\* @param requestEntity 请求头和请求体封装对象*
367. *\* @param responseType 返回对象类型*
368. *\* @param uriVariables URL中的变量，按顺序依次对应*
369. *\* @return ResponseEntity 响应对象封装类*
370. *\*/*
371. public <T> ResponseEntity<T> put(String url, HttpEntity<?> requestEntity, Class<T> responseType, Object... uriVariables) {
372. return restTemplate.exchange(url, HttpMethod.PUT, requestEntity, responseType, uriVariables);
373. }
375. */\*\**
376. *\* 自定义请求头和请求体的PUT请求调用方式*
377. *\**
378. *\* @param url 请求URL*
379. *\* @param requestEntity 请求头和请求体封装对象*
380. *\* @param responseType 返回对象类型*
381. *\* @param uriVariables URL中的变量，与Map中的key对应*
382. *\* @return ResponseEntity 响应对象封装类*
383. *\*/*
384. public <T> ResponseEntity<T> put(String url, HttpEntity<?> requestEntity, Class<T> responseType, Map<String, ?> uriVariables) {
385. return restTemplate.exchange(url, HttpMethod.PUT, requestEntity, responseType, uriVariables);
386. }
388. *// ----------------------------------DELETE-------------------------------------------------------*
390. */\*\**
391. *\* DELETE请求调用方式*
392. *\**
393. *\* @param url 请求URL*
394. *\* @param responseType 返回对象类型*
395. *\* @param uriVariables URL中的变量，按顺序依次对应*
396. *\* @return ResponseEntity 响应对象封装类*
397. *\*/*
398. public <T> ResponseEntity<T> delete(String url, Class<T> responseType, Object... uriVariables) {
399. return delete(url, HttpEntity.EMPTY, responseType, uriVariables);
400. }
402. */\*\**
403. *\* DELETE请求调用方式*
404. *\**
405. *\* @param url 请求URL*
406. *\* @param responseType 返回对象类型*
407. *\* @param uriVariables URL中的变量，与Map中的key对应*
408. *\* @return ResponseEntity 响应对象封装类*
409. *\*/*
410. public <T> ResponseEntity<T> delete(String url, Class<T> responseType, Map<String, ?> uriVariables) {
411. return delete(url, HttpEntity.EMPTY, responseType, uriVariables);
412. }
414. */\*\**
415. *\* DELETE请求调用方式*
416. *\**
417. *\* @param url 请求URL*
418. *\* @param requestBody 请求参数体*
419. *\* @param responseType 返回对象类型*
420. *\* @param uriVariables URL中的变量，按顺序依次对应*
421. *\* @return ResponseEntity 响应对象封装类*
422. *\*/*
423. public <T> ResponseEntity<T> delete(String url, Object requestBody, Class<T> responseType, Object... uriVariables) {
424. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody);
425. return delete(url, requestEntity, responseType, uriVariables);
426. }
428. */\*\**
429. *\* DELETE请求调用方式*
430. *\**
431. *\* @param url 请求URL*
432. *\* @param requestBody 请求参数体*
433. *\* @param responseType 返回对象类型*
434. *\* @param uriVariables URL中的变量，与Map中的key对应*
435. *\* @return ResponseEntity 响应对象封装类*
436. *\*/*
437. public <T> ResponseEntity<T> delete(String url, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
438. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody);
439. return delete(url, requestEntity, responseType, uriVariables);
440. }
442. */\*\**
443. *\* 带请求头的DELETE请求调用方式*
444. *\**
445. *\* @param url 请求URL*
446. *\* @param headers 请求头参数*
447. *\* @param responseType 返回对象类型*
448. *\* @param uriVariables URL中的变量，按顺序依次对应*
449. *\* @return ResponseEntity 响应对象封装类*
450. *\*/*
451. public <T> ResponseEntity<T> delete(String url, Map<String, String> headers, Class<T> responseType, Object... uriVariables) {
452. HttpHeaders httpHeaders = new HttpHeaders();
453. httpHeaders.setAll(headers);
454. return delete(url, httpHeaders, responseType, uriVariables);
455. }
457. */\*\**
458. *\* 带请求头的DELETE请求调用方式*
459. *\**
460. *\* @param url 请求URL*
461. *\* @param headers 请求头参数*
462. *\* @param responseType 返回对象类型*
463. *\* @param uriVariables URL中的变量，按顺序依次对应*
464. *\* @return ResponseEntity 响应对象封装类*
465. *\*/*
466. public <T> ResponseEntity<T> delete(String url, HttpHeaders headers, Class<T> responseType, Object... uriVariables) {
467. HttpEntity<Object> requestEntity = new HttpEntity<Object>(headers);
468. return delete(url, requestEntity, responseType, uriVariables);
469. }
471. */\*\**
472. *\* 带请求头的DELETE请求调用方式*
473. *\**
474. *\* @param url 请求URL*
475. *\* @param headers 请求头参数*
476. *\* @param responseType 返回对象类型*
477. *\* @param uriVariables URL中的变量，与Map中的key对应*
478. *\* @return ResponseEntity 响应对象封装类*
479. *\*/*
480. public <T> ResponseEntity<T> delete(String url, Map<String, String> headers, Class<T> responseType, Map<String, ?> uriVariables) {
481. HttpHeaders httpHeaders = new HttpHeaders();
482. httpHeaders.setAll(headers);
483. return delete(url, httpHeaders, responseType, uriVariables);
484. }
486. */\*\**
487. *\* 带请求头的DELETE请求调用方式*
488. *\**
489. *\* @param url 请求URL*
490. *\* @param headers 请求头参数*
491. *\* @param responseType 返回对象类型*
492. *\* @param uriVariables URL中的变量，与Map中的key对应*
493. *\* @return ResponseEntity 响应对象封装类*
494. *\*/*
495. public <T> ResponseEntity<T> delete(String url, HttpHeaders headers, Class<T> responseType, Map<String, ?> uriVariables) {
496. HttpEntity<Object> requestEntity = new HttpEntity<Object>(headers);
497. return delete(url, requestEntity, responseType, uriVariables);
498. }
500. */\*\**
501. *\* 带请求头的DELETE请求调用方式*
502. *\**
503. *\* @param url 请求URL*
504. *\* @param headers 请求头参数*
505. *\* @param requestBody 请求参数体*
506. *\* @param responseType 返回对象类型*
507. *\* @param uriVariables URL中的变量，按顺序依次对应*
508. *\* @return ResponseEntity 响应对象封装类*
509. *\*/*
510. public <T> ResponseEntity<T> delete(String url, Map<String, String> headers, Object requestBody, Class<T> responseType, Object... uriVariables) {
511. HttpHeaders httpHeaders = new HttpHeaders();
512. httpHeaders.setAll(headers);
513. return delete(url, httpHeaders, requestBody, responseType, uriVariables);
514. }
516. */\*\**
517. *\* 带请求头的DELETE请求调用方式*
518. *\**
519. *\* @param url 请求URL*
520. *\* @param headers 请求头参数*
521. *\* @param requestBody 请求参数体*
522. *\* @param responseType 返回对象类型*
523. *\* @param uriVariables URL中的变量，按顺序依次对应*
524. *\* @return ResponseEntity 响应对象封装类*
525. *\*/*
526. public <T> ResponseEntity<T> delete(String url, HttpHeaders headers, Object requestBody, Class<T> responseType, Object... uriVariables) {
527. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody, headers);
528. return delete(url, requestEntity, responseType, uriVariables);
529. }
531. */\*\**
532. *\* 带请求头的DELETE请求调用方式*
533. *\**
534. *\* @param url 请求URL*
535. *\* @param headers 请求头参数*
536. *\* @param requestBody 请求参数体*
537. *\* @param responseType 返回对象类型*
538. *\* @param uriVariables URL中的变量，与Map中的key对应*
539. *\* @return ResponseEntity 响应对象封装类*
540. *\*/*
541. public <T> ResponseEntity<T> delete(String url, Map<String, String> headers, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
542. HttpHeaders httpHeaders = new HttpHeaders();
543. httpHeaders.setAll(headers);
544. return delete(url, httpHeaders, requestBody, responseType, uriVariables);
545. }
547. */\*\**
548. *\* 带请求头的DELETE请求调用方式*
549. *\**
550. *\* @param url 请求URL*
551. *\* @param headers 请求头参数*
552. *\* @param requestBody 请求参数体*
553. *\* @param responseType 返回对象类型*
554. *\* @param uriVariables URL中的变量，与Map中的key对应*
555. *\* @return ResponseEntity 响应对象封装类*
556. *\*/*
557. public <T> ResponseEntity<T> delete(String url, HttpHeaders headers, Object requestBody, Class<T> responseType, Map<String, ?> uriVariables) {
558. HttpEntity<Object> requestEntity = new HttpEntity<Object>(requestBody, headers);
559. return delete(url, requestEntity, responseType, uriVariables);
560. }
562. */\*\**
563. *\* 自定义请求头和请求体的DELETE请求调用方式*
564. *\**
565. *\* @param url 请求URL*
566. *\* @param requestEntity 请求头和请求体封装对象*
567. *\* @param responseType 返回对象类型*
568. *\* @param uriVariables URL中的变量，按顺序依次对应*
569. *\* @return ResponseEntity 响应对象封装类*
570. *\*/*
571. public <T> ResponseEntity<T> delete(String url, HttpEntity<?> requestEntity, Class<T> responseType, Object... uriVariables) {
572. return restTemplate.exchange(url, HttpMethod.DELETE, requestEntity, responseType, uriVariables);
573. }
575. */\*\**
576. *\* 自定义请求头和请求体的DELETE请求调用方式*
577. *\**
578. *\* @param url 请求URL*
579. *\* @param requestEntity 请求头和请求体封装对象*
580. *\* @param responseType 返回对象类型*
581. *\* @param uriVariables URL中的变量，与Map中的key对应*
582. *\* @return ResponseEntity 响应对象封装类*
583. *\*/*
584. public <T> ResponseEntity<T> delete(String url, HttpEntity<?> requestEntity, Class<T> responseType, Map<String, ?> uriVariables) {
585. return restTemplate.exchange(url, HttpMethod.DELETE, requestEntity, responseType, uriVariables);
586. }
588. *// ----------------------------------通用方法-------------------------------------------------------*
590. */\*\**
591. *\* 通用调用方式*
592. *\**
593. *\* @param url 请求URL*
594. *\* @param method 请求方法类型*
595. *\* @param requestEntity 请求头和请求体封装对象*
596. *\* @param responseType 返回对象类型*
597. *\* @param uriVariables URL中的变量，按顺序依次对应*
598. *\* @return ResponseEntity 响应对象封装类*
599. *\*/*
600. public <T> ResponseEntity<T> exchange(String url, HttpMethod method, HttpEntity<?> requestEntity, Class<T> responseType, Object... uriVariables) {
601. return restTemplate.exchange(url, method, requestEntity, responseType, uriVariables);
602. }
604. */\*\**
605. *\* 通用调用方式*
606. *\**
607. *\* @param url 请求URL*
608. *\* @param method 请求方法类型*
609. *\* @param requestEntity 请求头和请求体封装对象*
610. *\* @param responseType 返回对象类型*
611. *\* @param uriVariables URL中的变量，与Map中的key对应*
612. *\* @return ResponseEntity 响应对象封装类*
613. *\*/*
614. public <T> ResponseEntity<T> exchange(String url, HttpMethod method, HttpEntity<?> requestEntity, Class<T> responseType, Map<String, ?> uriVariables) {
615. return restTemplate.exchange(url, method, requestEntity, responseType, uriVariables);
616. }
617. }

**4.controller层+service层实现**

1. package com.aircas.satellitemanagement.httpclient;

4. *//import com.aircas.satellitemanagement.exception.success.WebAPIResponse1;*
6. import com.aircas.satellitemanagement.jpa.User;
7. import com.aircas.satellitemanagement.jpa.UserService;
8. import org.springframework.beans.factory.annotation.Autowired;
9. import org.springframework.data.domain.Page;
10. import org.springframework.web.bind.annotation.GetMapping;
11. import org.springframework.web.bind.annotation.RequestMapping;
12. import org.springframework.web.bind.annotation.RequestParam;
13. import org.springframework.web.bind.annotation.RestController;
15. import java.util.List;
17. @RestController
18. @RequestMapping(value = "/http")
19. public class HttpController {
21. @Autowired
22. HttpService httpService;
24. @GetMapping(value = "/1")
25. public String get() {
26. String s = httpService.get();
27. return s;
28. }

31. }
32. package com.aircas.satellitemanagement.httpclient;
34. import com.aircas.satellitemanagement.jpa.User;
35. import com.aircas.satellitemanagement.jpa.UserRepository;
36. import lombok.extern.slf4j.Slf4j;
37. import org.springframework.beans.factory.annotation.Autowired;
38. import org.springframework.data.domain.Page;
39. import org.springframework.data.domain.PageRequest;
40. import org.springframework.data.domain.Pageable;
41. import org.springframework.http.ResponseEntity;
42. import org.springframework.stereotype.Service;
44. import java.util.List;
46. @Service
47. @Slf4j
48. public class HttpService {
50. @Autowired
51. RestTemplateUtil restTemplateUtil;
53. public String get() {
54. ResponseEntity<String> stringResponseEntity = restTemplateUtil.get("http://localhost:9101/user/all/3", String.class);
55. String body = stringResponseEntity.getBody();
56. System.out.println(body);
57. return body;
58. }

61. }

**5.post请求可能会出现的问题以及解决办法：**

最近使用[RestTemplate发送post请求](https://so.csdn.net/so/search?q=RestTemplate%E5%8F%91%E9%80%81post%E8%AF%B7%E6%B1%82&spm=1001.2101.3001.7020)，遇到了很多问题，如转换httpMessage失败、中文乱码等，调了好久才找到下面较为简便的方法：

1. RestTemplate restTemplate = new RestTemplate();
2. HttpHeaders headers = new HttpHeaders();
3. MediaType type = MediaType.parseMediaType("application/json; charset=UTF-8");
4. headers.setContentType(type);
5. headers.add("Accept", MediaType.APPLICATION\_JSON.toString());
7. JSONObject jsonObj = JSONObject.fromObject(params);
9. HttpEntity<String> formEntity = new HttpEntity<String>(jsonObj.toString(), headers);
11. String result = restTemplate.postForObject(url, formEntity, String.class);

如果直接使用在postForObject中把对象传入很容易出现no suitable HttpMessageConverter found for request type的错误，建议直接先转成字符串，见jsonObj.otString()，网上有人说设置[RestTemplate](https://so.csdn.net/so/search?q=RestTemplate&spm=1001.2101.3001.7020" \t "_blank)的HttpMessageConverter，试了一下要引入各种包。另外要注意中文编码问题，网上有人说StringHttpMessageConverter默认使用ISO-8859-1，要指定为UTF-8编码，自己尝试没有成功，最后通过指定contentType的方式解决了。