

[🏠 / 开发指南 / 后端手册](#)[👤 芋道源码](#) [📅 2022-04-03](#)

Redis 缓存

[yudao-spring-boot-starter-redis](#) [🔗](#) 技术组件，使用 Redis 实现缓存的功能，它有 2 种使用方式：

- 编程式缓存：基于 Spring Data Redis 框架的 RedisTemplate 操作模板
- 声明式缓存：基于 Spring Cache 框架的 [@Cacheable](#) 等等注解

1. 编程式缓存

友情提示：

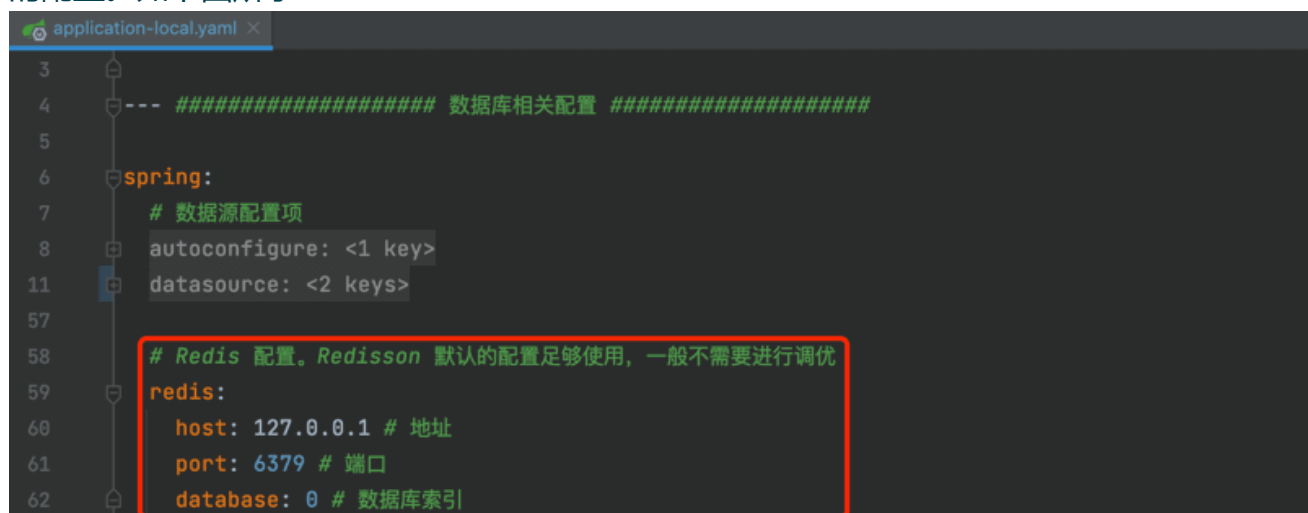
如果你未学习过 Spring Data Redis 框架，可以后续阅读 [《芋道 Spring Boot Redis 入门》](#) [🔗](#) 文章。

```
<dependency>
  <groupId>org.redisson</groupId>
  <artifactId>redisson-spring-boot-starter</artifactId>
</dependency>
```

由于 Redisson 提供了分布式锁、队列、限流等特性，所以使用它作为 Spring Data Redis 的客户端。

1.1 Spring Data Redis 配置

① 在 [application-local.yaml](#) [🔗](#) 配置文件中，通过 [spring.redis](#) 配置项，设置 Redis 的配置。如下图所示：



```
application-local.yaml
3
4  --- ##### 数据库相关配置 #####
5
6  spring:
7    # 数据源配置项
8    autoconfigure: <1 key>
11   datasource: <2 keys>
57
58   # Redis 配置。Redisson 默认的配置足够使用，一般不需要进行调优
59   redis:
60     host: 127.0.0.1 # 地址
61     port: 6379 # 端口
62     database: 0 # 数据库索引
```

② 在 `YudaoRedisAutoConfiguration` 配置类，设置使用 JSON 序列化 value 值。如下图所示：

```
@Configuration
public class YudaoRedisAutoConfiguration {

    /**
     * 创建 RedisTemplate Bean, 使用 JSON 序列化方式
     */
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
        // 创建 RedisTemplate 对象
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        // 设置 RedisConnectionFactory 工厂。它实现多种 Java Redis 客户端接入的秘密工厂。感兴趣的朋友，可以自己去撸下。
        template.setConnectionFactory(factory);
        // 使用 String 序列化方式，序列化 KEY 。
        template.setKeySerializer(RedisSerializer.string());
        template.setHashKeySerializer(RedisSerializer.string());
        // 使用 JSON 序列化方式（库是 Jackson），序列化 VALUE 。
        template.setValueSerializer(RedisSerializer.json());
        template.setHashValueSerializer(RedisSerializer.json());
        return template;
    }
}
```

1.2 实战案例

以访问令牌 Access Token 的缓存来举例子，讲解项目中是如何使用 Spring Data Redis 框架的。

```
127.0.0.1:6379> get oauth2_access_token:f440d0f6205344d18a6dcfce51dcacbb
"{\"createTime\":null,\"updateTime\":null,\"creator\":null,\"updater\":null,\"deleted\":null,\"tenantId\":1,\"id\":1696,\"accessToken\":\"f440d0f6205344d18a6dcfce51dcacbb\",\"refreshToken\":\"85a0761f22eb4adf8284c7e6151fc303\", \"userId\":1,\"userType\":2,\"clientId\":\"default\",\"scopes\":null,\"expiresTime\":1677822301000}"
```

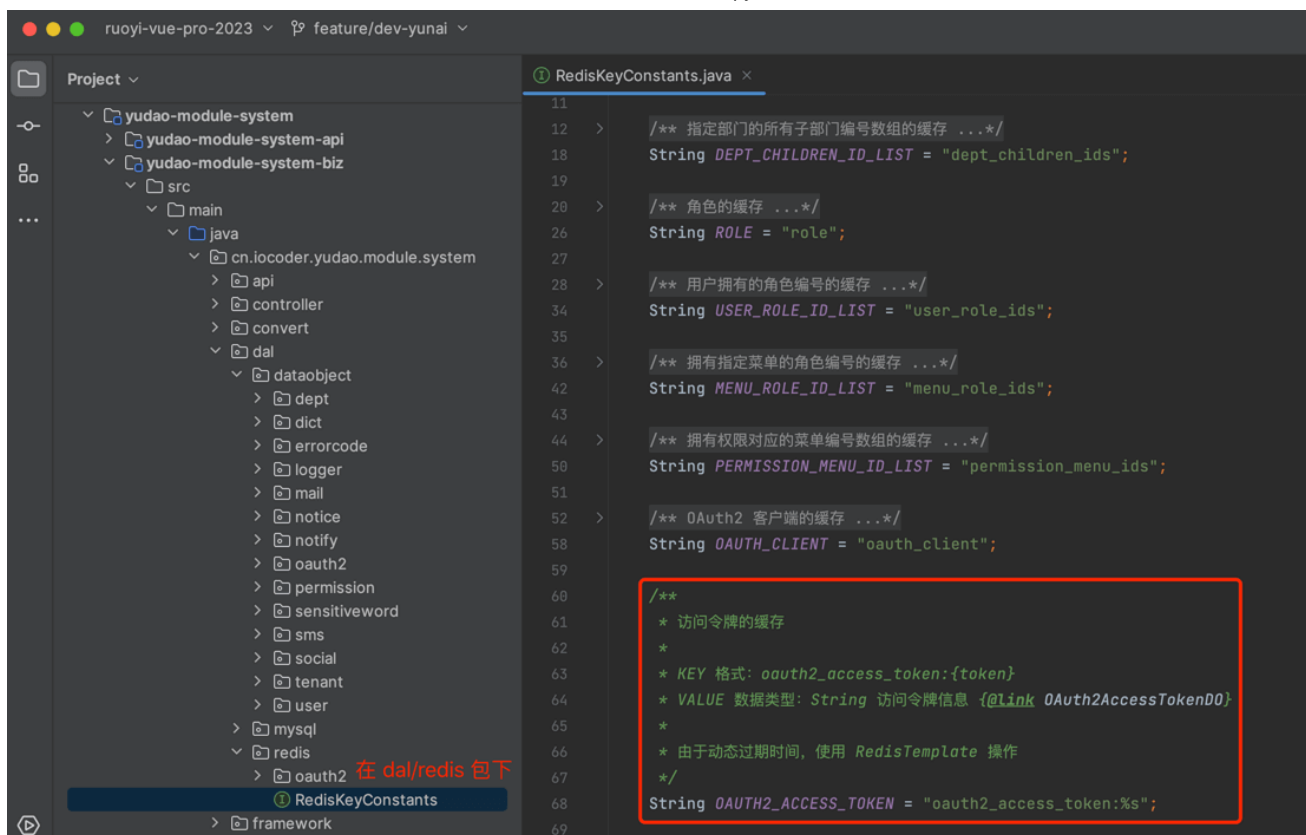
1.2.1 引入依赖

在 `yudao-module-system-biz` 模块中，引入 `yudao-spring-boot-starter-redis` 技术组件。如下所示：

```
<dependency>
    <groupId>cn.iocoder.boot</groupId>
    <artifactId>yudao-spring-boot-starter-redis</artifactId>
</dependency>
```

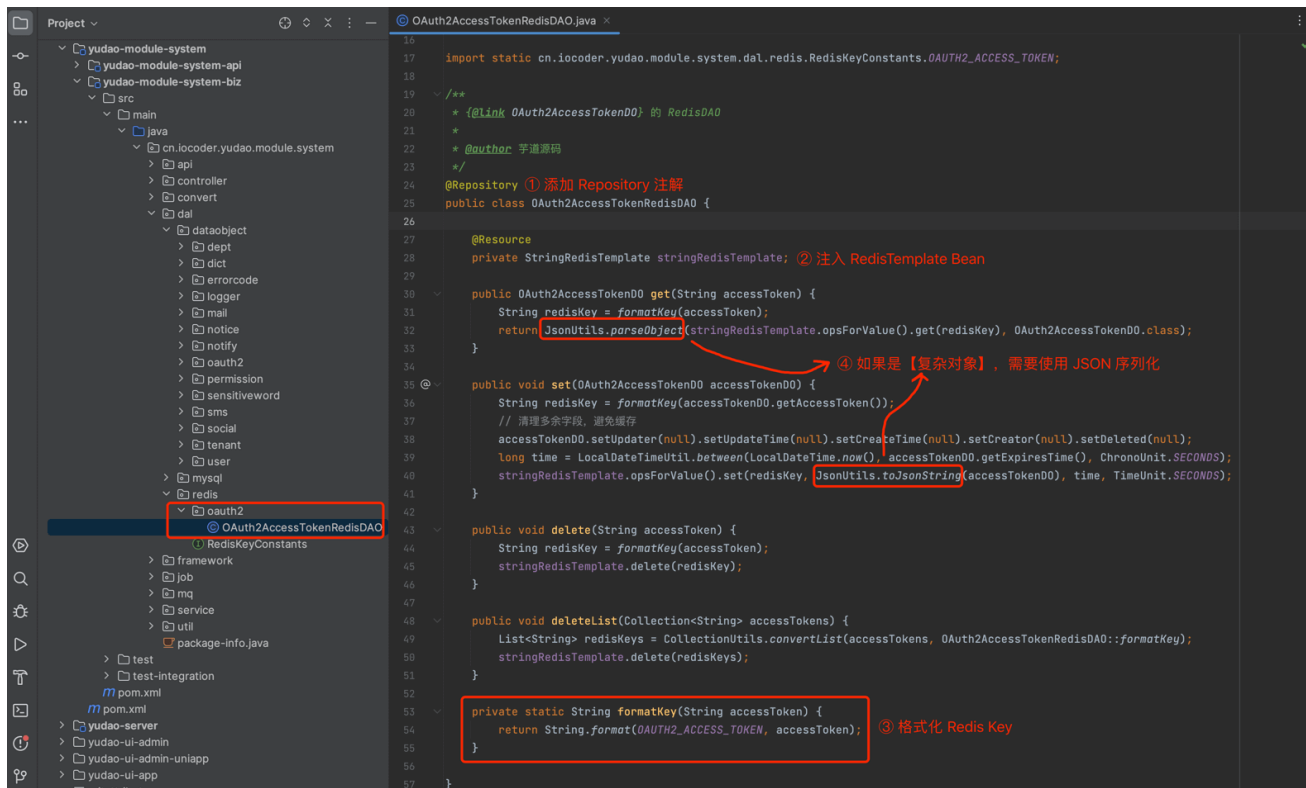
1.2.2 OAuth2AccessTokenDO

新建 `OAuth2AccessTokenDO` 类，访问令牌 Access Token 类。代码如下：



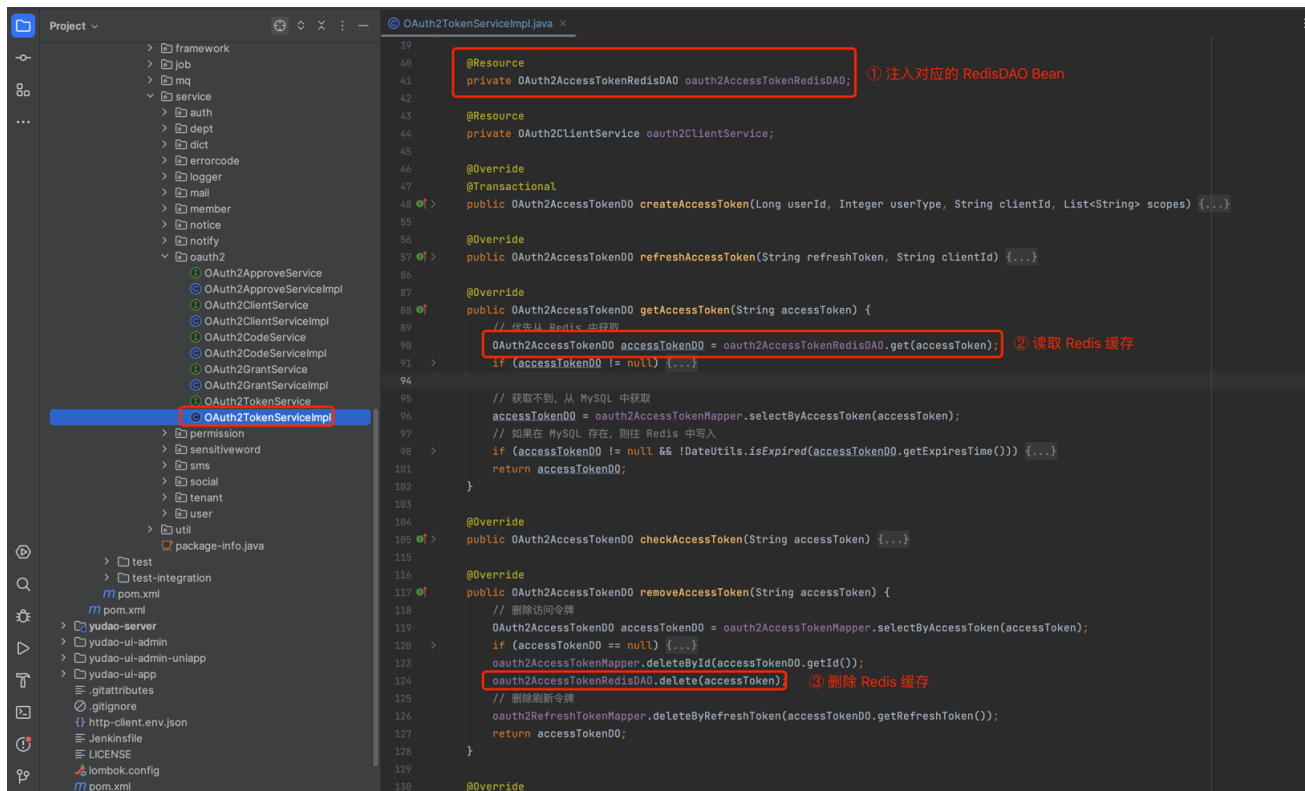
1.2.4 OAuth2AccessTokenRedisDAO

新建 `OAuth2AccessTokenRedisDAO` 类, 是 `OAuth2AccessTokenDO` 的 RedisDAO 实现。代码如下:



1.2.5 OAuth2TokenServiceImpl

在 `OAuth2TokenServiceImpl` 中, 只要注入 `OAuth2AccessTokenRedisDAO` Bean, 非常简洁干净的进行 `OAuth2AccessTokenDO` 的缓存操作, 无需关心具体的实现。代码如下:



2. 声明式缓存

友情提示:

如果你未学习过 Spring Cache 框架，可以后续阅读 [《芋道 Spring Boot Cache 入门》](#) 文章。

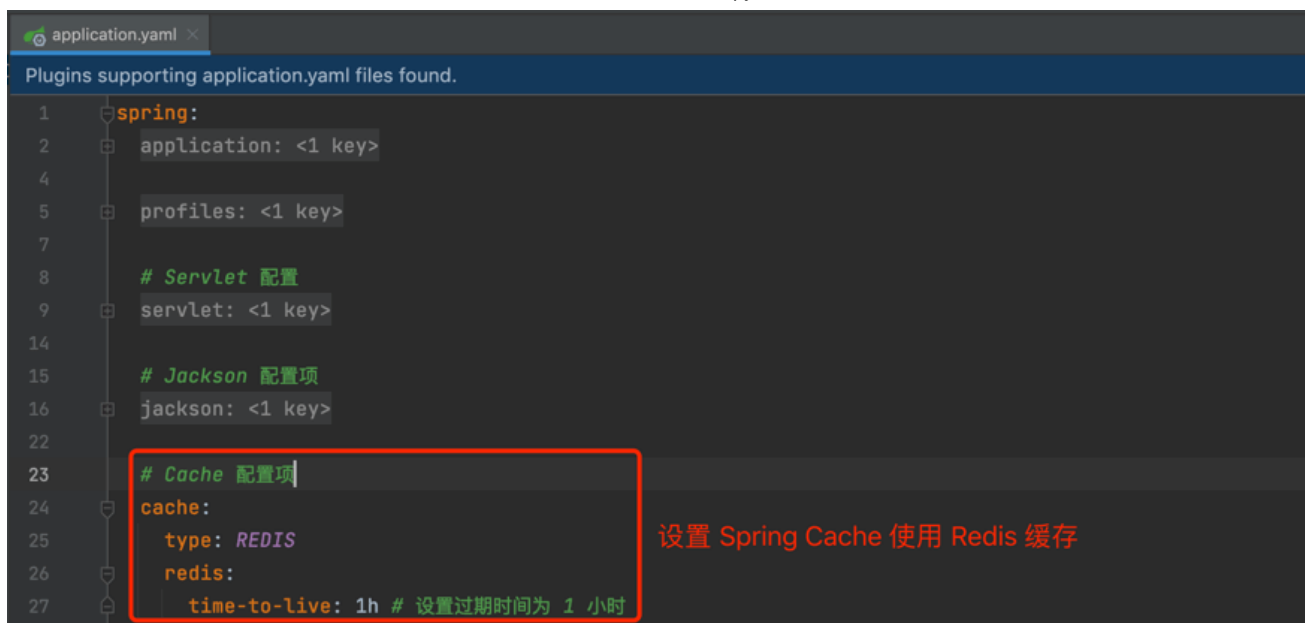
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

相比来说 Spring Data Redis 编程式缓存，Spring Cache 声明式缓存的使用更加便利，一个 `@Cacheable` 注解即可实现缓存的功能。示例如下：

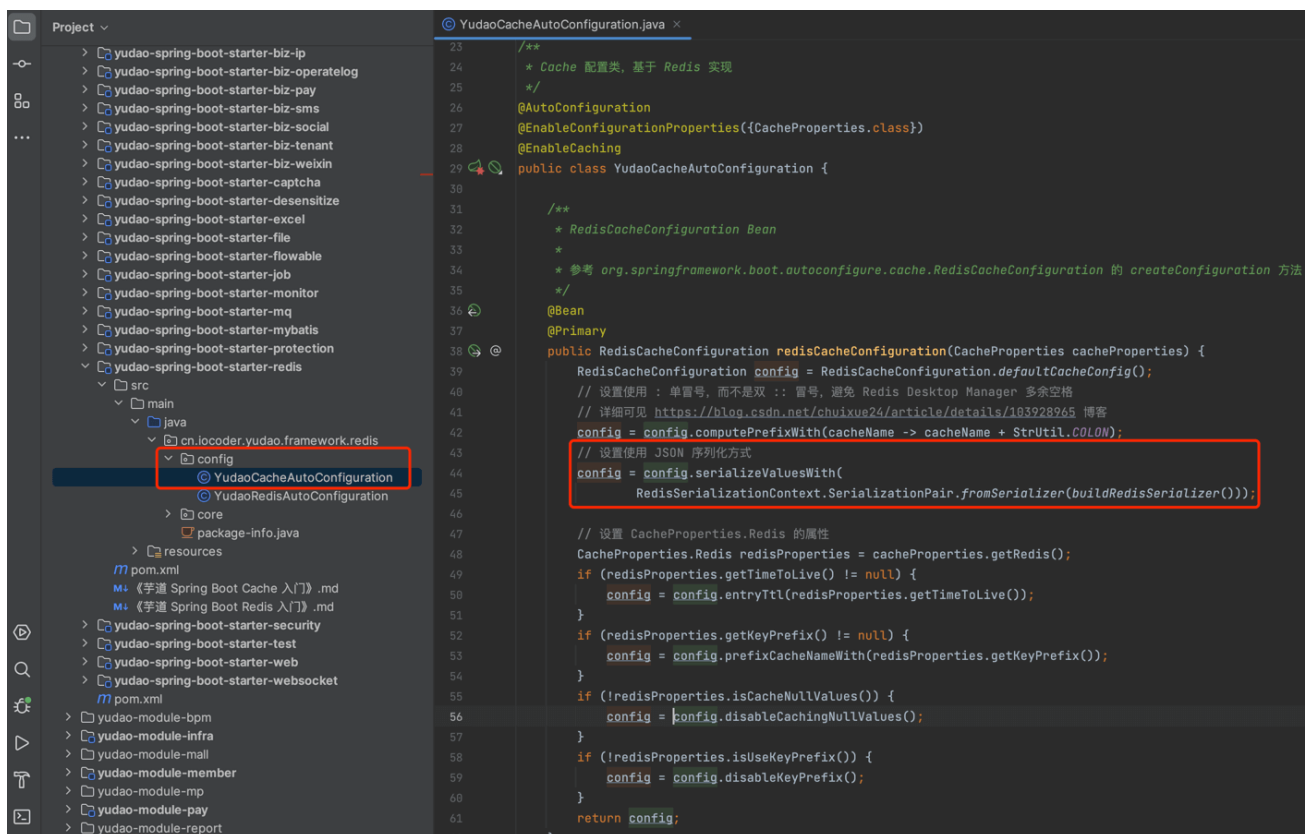
```
@Cacheable(value = "users", key = "#id")
UserDO getUserById(Integer id);
```

2.1 Spring Cache 配置

① 在 `application.yaml` 配置文件中，通过 `spring.redis` 配置项，设置 Redis 的配置。如下图所示：



② 在 `YudaoCacheAutoConfiguration` 配置类，设置使用 JSON 序列化 value 值。如下图所示：



2.2 常见注解

2.2.1 @Cacheable 注解

`@Cacheable` 注解：添加在方法上，缓存方法的执行结果。执行过程如下：

- 1) 首先，判断方法执行结果的缓存。如果有，则直接返回该缓存结果。
- 2) 然后，执行方法，获得方法结果。
- 3) 之后，根据是否满足缓存的条件。如果满足，则缓存方法结果到缓存。
- 4) 最后，返回方法结果。

2.2.2 @CachePut 注解

`@CachePut` 注解，添加在方法上，缓存方法的执行结果。不同于 `@Cacheable` 注解，它的执行过程如下：

- 1) 首先，执行方法，获得方法结果。也就是说，无论是否有缓存，都会执行方法。
- 2) 然后，根据是否满足缓存的条件。如果满足，则缓存方法结果到缓存。
- 3) 最后，返回方法结果。

2.2.3 @CacheEvict 注解

`@CacheEvict` 注解，添加在方法上，删除缓存。

2.3 实战案例

在 `RoleServiceImpl` 中，使用 Spring Cache 实现了 Role 角色缓存，采用【被动读】的方案。原因是：

```
37
38  /**
39   * 角色 Service 实现类
40   *
41   * @author 李道源码
42   */
43  @Service
44  @Slf4j
45  @TenantDS
46  public class RoleServiceImpl implements RoleService {
47
48      @Resource
49      private PermissionService permissionService;
50
51      @Resource
52      private RoleMapper roleMapper;
53
54      @Override
55      @Cacheable(value = RedisKeyConstants.ROLE, key = "#id",
56               unless = "#result == null") ① 读取时，写入缓存
57      public RoleDO getRoleFromCache(Long id) { return roleMapper.selectById(id); }
58
59
60      @Override
61      @Transactional(rollbackFor = Exception.class)
62      public Long createRole(RoleCreateReqVO reqVO, Integer type) {...} ② 创建时，无需操作缓存
63
64
65      @Override
66      @CacheEvict(value = RedisKeyConstants.ROLE, key = "#reqVO.id") ③ 更新时，删除缓存
67      public void updateRole(RoleUpdateReqVO reqVO) {...}
68
69
70      @Override
71      @CacheEvict(value = RedisKeyConstants.ROLE, key = "#id")
72      public void updateRoleStatus(Long id, Integer status) {...}
73
74
75      @Override
76      @CacheEvict(value = RedisKeyConstants.ROLE, key = "#id")
77      public void updateRoleDataScope(Long id, Integer dataScope, Set<Long> dataScopeDeptIds) {...}
78
79
80      @Override
81      @Transactional(rollbackFor = Exception.class)
82      @CacheEvict(value = RedisKeyConstants.ROLE, key = "#id")
83      public void deleteRole(Long id) {...}
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
```

- 【被动读】相对能够保证 Redis 与 MySQL 的一致性
- 绝大多数数据不需要放到 Redis 缓存中，采用【主动写】会将非必要的数据进行缓存

友情提示：

如果你未学习过 MySQL 与 Redis 一致性的问题，可以后续阅读 [《Redis 与 MySQL 双写一致性如何保证？》](#) 文章。

① 执行 `#getRoleFromCache(...)` 方法，从 MySQL 读取数据后，向 Redis 写入缓存。如下图所示：

```
127.0.0.1:6379> keys role*
1) "role:t1:1"
2) "role:t1:2"
127.0.0.1:6379> get role:t1:1
"{@class\":\"cn.iocoder.yudao.module.system.dal.dataobject.permission.RoleDO\",\"createTime\":[2021,1,5,17,3,48],\"updateTime\":[2022,2,22,5,8,21],\"creator\":\"admin\",\"updater\":\"\",\"deleted\":false,\"tenantId\":1,\"id\":1,\"name\":\"\\xe8\\xb6\\x85\\xe7\\xba\\xa7\\xe7\\xae\\xa1\\xe7\\x90\\x86\\xe5\\x91\\x98\",\"code\":\"super_admin\",\"sort\":1,\"status\":0,\"type\":1,\"remark\":\"\\xe8\\xb6\\x85\\xe7\\xba\\xa7\\xe7\\xae\\xa1\\xe7\\x90\\x86\\xe5\\x91\\x98\",\"dataScope\":1,\"dataScopeDeptIds\":null}"
```

其中 t1 指的租户 1，可忽略。
1 和 2 是角色编号

② 执行 `#updateRole(...)` 或 `#deleteRole(...)` 方法，在更新或者删除 MySQL 数据后，从 Redis 删除缓存。如下图所示：

```
127.0.0.1:6379> get role:t1:1
(nil)
127.0.0.1:6379> 
```

被删除，无法查询到

2.4 过期时间

Spring Cache 默认使用 `spring.cache.redis.time-to-live` 配置项，设置缓存的过期时间，项目默认为 1 小时。

如果你想自定义过期时间，可以在 `@Cacheable` 注解中的 `cacheNames` 属性中，添加 `#{过期时间}` 后缀，单位是秒。如下图所示：

```
54  @Override
55  @Cacheable(value = RedisKeyConstants.ROLE + "#60", key = "#id",
56             unless = "#result == null")
57  public RoleDO getRoleFromCache(Long id) { return roleMapper.selectById(id); }
```

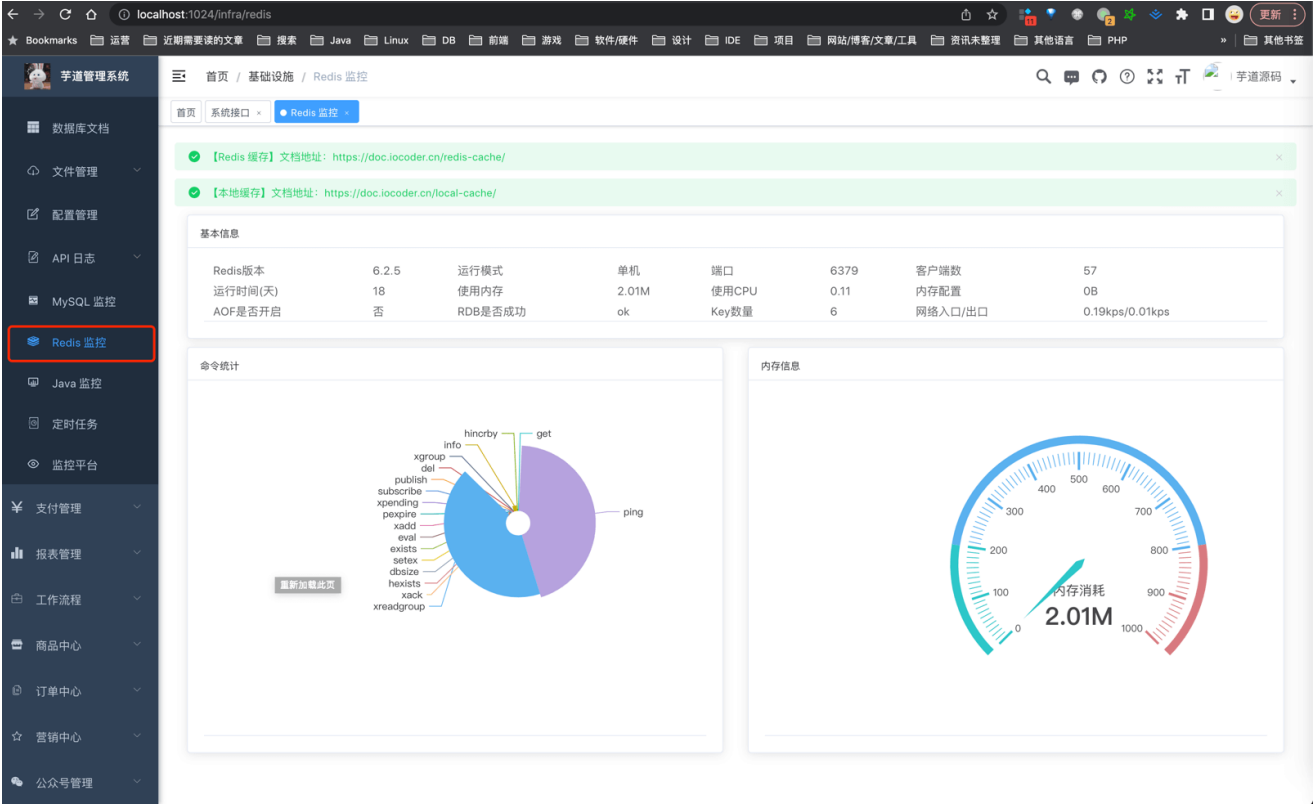
默认单位：秒

实现的原来，参考 [《Spring @Cacheable 扩展支持自定义过期时间》](#) 文章。

3. Redis 监控

`yudao-module-infra` 的 `redis` 模块，提供了 Redis 监控的功能。

点击 [基础设施 -> Redis 监控] 菜单，可以查看到 Redis 的基础信息、命令统计、内存信息。如下图所示：



← 多数据源（读写分离）

本地缓存→



Theme by Vdoing | Copyright © 2019-2023 芋道源码 | MIT License