

j2cache

介绍

j2cache入门案例

- 第一步：创建工程j2cache_demo
- 第二步：修改pom文件
- 第三步：修改application.yml文件
- 第四步：启动Redis
- 第五步：创建/resources/caffeine.properties文件
- 第六步：编写TestController
- 第七步：启动程序
- 第八步：访问

无法启动解决

测试缓存击穿

- 第一步：修改TestController
- 第二步：启动程序，并打开jmeter
- 第三步：设置http请求
- 第四步：添加监听器
- 第五步：启动jmeter
- 第六步：主动清除缓存

测试缓存穿透

- 第一步：修改TestController
- 第二步：设置http请求
- 第三步：重启服务并启动jmeter

自定义spring boot starter

开发starter

- 第一步：初始化项目
- 第二步：修改pom文件
- 第三步：修改类J2CacheCache
- 第四步：修改类J2CacheCacheManger
- 第五步：添加类J2CacheSerializer
- 第六步：添加类SpringJ2CacheConfigUtil
- 第七步：添加类SpringUtil
- 第八步：添加类ConfigureNotifyKeyspaceEventsAction
- 第九步：添加类SpringRedisActiveMessageListener
- 第十步：添加类SpringRedisCache
- 第十一步：添加类SpringRedisGenericCache
- 第十二步：添加类SpringRedisMessageListener
- 第十三步：添加类SpringRedisProvider
- 第十四步：添加类SpringRedisPubSubPolicy
- 第十五步：添加配置类J2CacheAutoConfiguration
- 第十六步：添加配置属性类J2CacheConfig
- 第十七步：添加配置类J2CacheSpringCacheAutoConfiguration
- 第十八步：添加配置类J2CacheSpringRedisAutoConfiguration
- 第十九步：添加配置类CacheConfig
- 第二十步：添加实体类RedisData
- 第二十一部：添加配置类RedissonConfig
- 第二十二步：添加工具类RedisUtils
- 第二十三步：添加配置类RedisUtilsConfig
- 第二十四步：编写spring.factories

使用starter

- 第一步：添加tools-j2cache的依赖
- 第二步：编写配置文件application.yml
- 第三步：编写实体类Student
- 第四步：编写TestController
- 第五步：启动程序

j2cache

介绍

j2cache是OSChina目前正在使用的两级缓存框架。

j2cache的两级缓存结构：

- L1： 进程内缓存 caffeine/ehcache
- L2： 集中式缓存 Redis/Memcached

j2cache其实并不是在重复造轮子，而是作资源整合，即将Ehcache、Caffeine、redis、Spring Cache等进行整合。

由于大量的缓存读取会导致L2的网络成为整个系统的瓶颈，因此L1的目标是降低对L2的读取次数。该缓存框架主要用于集群环境中。单机也可使用，用于避免应用重启导致的ehcache缓存数据丢失。

j2cache从1.3.0版本开始支持JGroups和Redis Pub/Sub两种方式进行缓存事件的通知。

数据读取顺序 -> L1 -> L2 -> DB

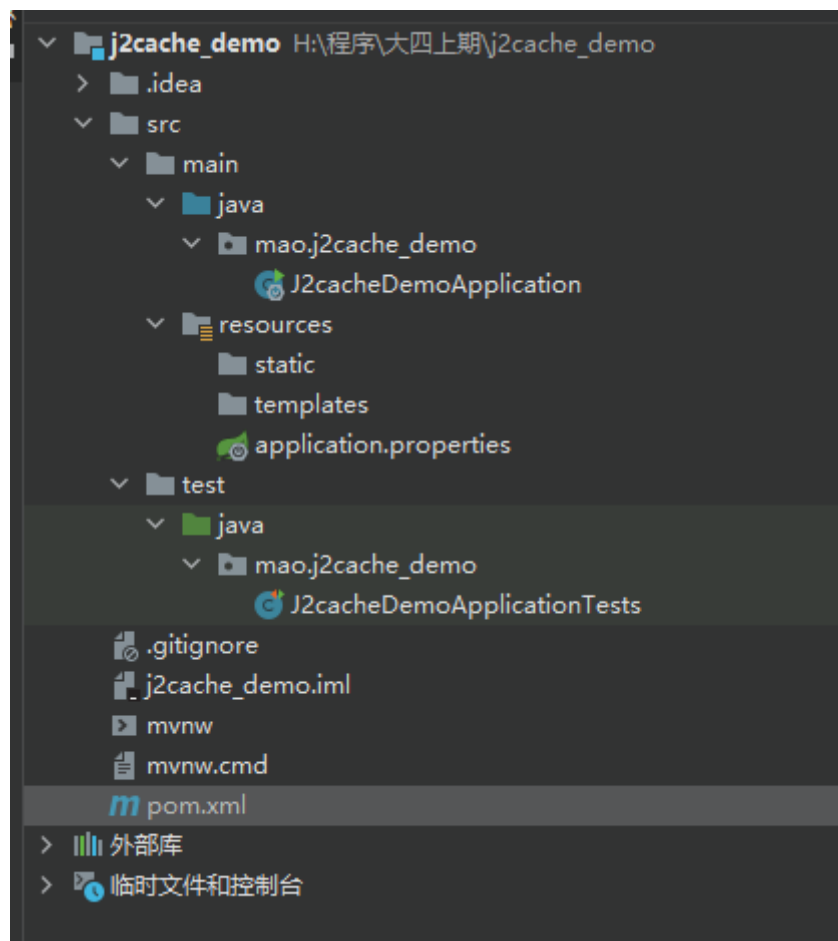
使用j2cache需要导入的maven坐标：

```
1 <dependency>
2   <groupId>net.oschina.j2cache</groupId>
3   <artifactId>j2cache-spring-boot2-starter</artifactId>
4   <version>2.8.0-release</version>
5 </dependency>
6 <dependency>
7   <groupId>net.oschina.j2cache</groupId>
```

```
8      <artifactId>j2cache-core</artifactId>
9      <version>2.8.0-release</version>
10     <exclusions>
11       <exclusion>
12         <groupId>org.slf4j</groupId>
13         <artifactId>slf4j-simple</artifactId>
14       </exclusion>
15       <exclusion>
16         <groupId>org.slf4j</groupId>
17         <artifactId>slf4j-api</artifactId>
18       </exclusion>
19     </exclusions>
20 </dependency>
```

j2cache入门案例

第一步：创建工程j2cache_demo



第二步：修改pom文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.7.1</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>mao</groupId>
12    <artifactId>j2cache_demo</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>j2cache_demo</name>
15    <description>j2cache_demo</description>
16    <properties>
17        <java.version>8</java.version>
18    </properties>
19    <dependencies>
20
21        <dependency>
```

```

22         <groupId>org.springframework.boot</groupId>
23         <artifactId>spring-boot-starter-web</artifactId>
24     </dependency>
25
26     <dependency>
27         <groupId>org.springframework.boot</groupId>
28         <artifactId>spring-boot-starter-test</artifactId>
29         <scope>test</scope>
30     </dependency>
31
32     <dependency>
33         <groupId>net.oschina.j2cache</groupId>
34         <artifactId>j2cache-spring-boot2-starter</artifactId>
35         <version>2.8.0-release</version>
36     </dependency>
37     <dependency>
38         <groupId>net.oschina.j2cache</groupId>
39         <artifactId>j2cache-core</artifactId>
40         <version>2.8.0-release</version>
41         <exclusions>
42             <exclusion>
43                 <groupId>org.slf4j</groupId>
44                 <artifactId>slf4j-simple</artifactId>
45             </exclusion>
46             <exclusion>
47                 <groupId>org.slf4j</groupId>
48                 <artifactId>slf4j-api</artifactId>
49             </exclusion>
50         </exclusions>
51     </dependency>
52
53 </dependencies>
54
55 <build>
56     <plugins>
57         <plugin>
58             <groupId>org.springframework.boot</groupId>
59             <artifactId>spring-boot-maven-plugin</artifactId>
60         </plugin>
61     </plugins>
62 </build>
63
64 </project>

```

第三步：修改application.yml文件

```

1  spring:
2    cache:
3      type: GENERIC
4    redis:

```

```

5     host: 127.0.0.1
6     password: 123456
7     port: 6379
8     database: 0
9
10    j2cache:
11        # config-location: /j2cache.properties
12        open-spring-cache: true
13        cache-clean-mode: passive
14        allow-null-values: true
15        redis-client: lettuce #指定redis客户端使用lettuce, 也可以使用Jedis
16        l2-cache-open: true #开启二级缓存
17        broadcast: net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
18        # broadcast: jgroups
19        L1: #指定一级缓存提供者caffeine
20            provider_class: caffeine
21        L2: #指定二级缓存提供者redis
22            provider_class:
23                net.oschina.j2cache.cache.support.redis.SpringRedisProvider
24            config_section: lettuce
25        sync_ttl_to_redis: true
26        default_cache_null_object: false
27        serialization: fst
28    caffeine:
29        properties: /caffeine.properties # 这个配置文件需要放在项目中
30    lettuce:
31        mode: single
32        namespace:
33        storage: generic
34        channel: j2cache
35        scheme: redis
36        hosts: 127.0.0.1:6379
37        password: 123456
38        database: 0
39        sentinelMasterId:
40        maxTotal: 100
41        maxIdle: 10
42        minIdle: 10
43        timeout: 10000

```

第四步：启动Redis

```

1 C:\Users\mao>redis-cli
2 127.0.0.1:6379> auth 123456
3 OK
4 127.0.0.1:6379> ping
5 PONG
6 127.0.0.1:6379>

```

第五步：创建/resources/caffeine.properties文件

```
1 #####
2 # Caffeine configuration
3 # [name] = size, xxxx[s|m|h|d]
4 #####
5 default=2000, 2h
6 rx=50, 2h
```

第六步：编写TestController

```
1 package mao.j2cache_demo.controller;
2
3 import net.oschina.j2cache.CacheChannel;
4 import net.oschina.j2cache.CacheObject;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 import java.util.ArrayList;
10 import java.util.List;
11
12 /**
13  * Project name(项目名称): j2cache_demo
14  * Package(包名): mao.j2cache_demo.controller
15  * Class(类名): TestController
16  * Author(作者): mao
17  * Author QQ: 1296193245
18  * GitHub: https://github.com/maomao124/
19  * Date(创建日期): 2022/11/5
20  * Time(创建时间): 13:22
21  * Version(版本): 1.0
22  * Description(描述): 无
23  */
24
25 @RestController
26 public class TestController
27 {
28
29     @Autowired
30     private CacheChannel cacheChannel;
31
32     private final String key = "myKey";
33     private final String region = "rx";
34 }
```

```

35
36 @GetMapping("/getInfos")
37 public List<String> getInfos()
38 {
39     CacheObject cacheObject = cacheChannel.get(region, key);
40     if (cacheObject.getValue() == null)
41     {
42         //缓存中没有找到，查询数据库获得
43         List<String> data = new ArrayList<>();
44         data.add("info1");
45         data.add("info2");
46         //放入缓存
47         cacheChannel.set(region, key, data);
48         return data;
49     }
50     return (List<String>) cacheObject.getValue();
51 }
52
53 /**
54  * 清理指定缓存
55  *
56  * @return {@link String}
57  */
58 @GetMapping("/evict")
59 public String evict()
60 {
61     cacheChannel.evict(region, key);
62     return "evict success";
63 }
64
65 /**
66  * 检测存在那级缓存
67  *
68  * @return {@link String}
69  */
70 @GetMapping("/check")
71 public String check()
72 {
73     int check = cacheChannel.check(region, key);
74     return "level:" + check;
75 }
76
77 /**
78  * 检测缓存数据是否存在
79  *
80  * @return {@link String}
81  */
82 @GetMapping("/exists")
83 public String exists()
84 {
85     boolean exists = cacheChannel.exists(region, key);
86     return "exists:" + exists;
87 }
88
89 /**
90  * 清理指定区域的缓存
91  *
92  * @return {@link String}

```



```
93     */
94     @GetMapping("/clear")
95     public String clear()
96     {
97         cacheChannel.clear(region);
98         return "clear success";
99     }
100 }
```

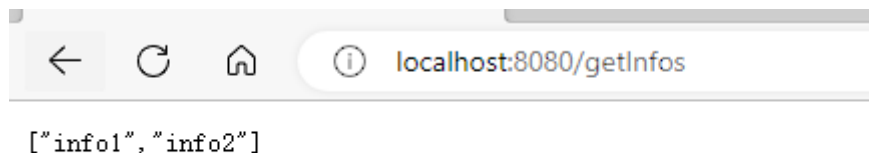
第七步：启动程序

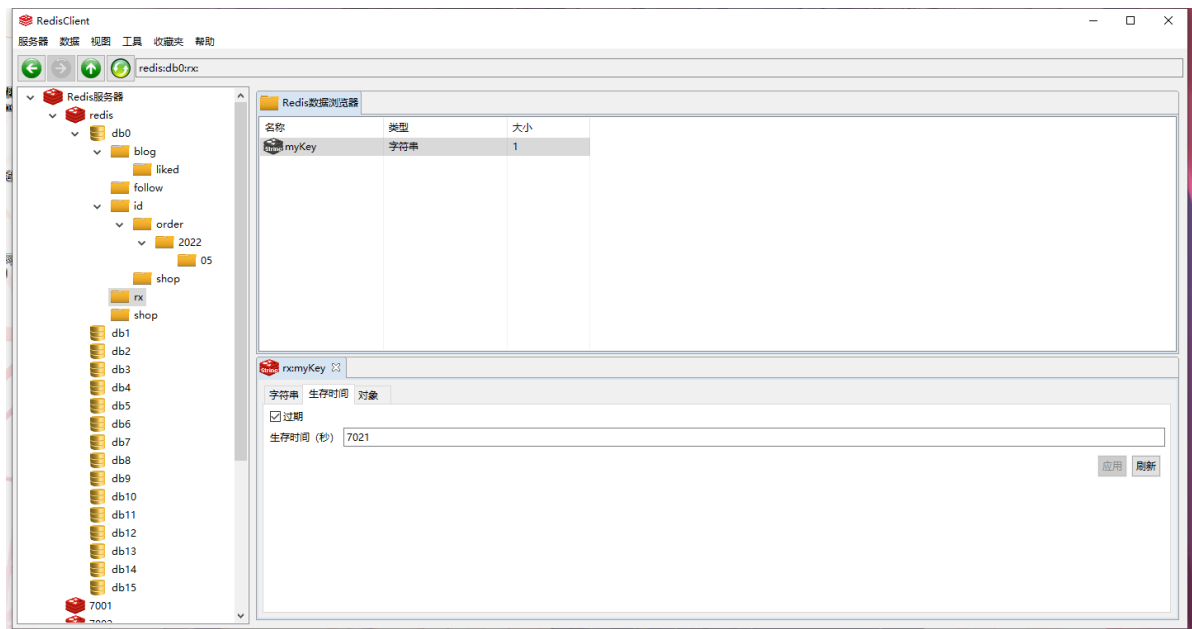
[illegible]

```
20 2022-11-05 13:43:47.580 INFO 10704 --- [          main]
    w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext:
    initialization completed in 918 ms
21 2022-11-05 13:43:47.671 INFO 10704 --- [          main]
    n.o.j2cache.util.SerializationUtils      : Using Serializer ->
    [fst:net.oschina.j2cache.util.FSTSerializer]
22 2022-11-05 13:43:47.674 INFO 10704 --- [          main]
    net.oschina.j2cache.CacheProviderHolder  : Using L1 CacheProvider :
    net.oschina.j2cache.caffeine.CaffeineProvider
23 2022-11-05 13:43:47.864 INFO 10704 --- [          main]
    net.oschina.j2cache.CacheProviderHolder  : Using L2 CacheProvider :
    net.oschina.j2cache.cache.support.redis.SpringRedisProvider
24 2022-11-05 13:43:47.873 INFO 10704 --- [          main]
    net.oschina.j2cache.J2CacheBuilder       : Using cluster policy :
    net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
25 2022-11-05 13:43:48.231 INFO 10704 --- [          main]
    o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080
    (http) with context path ''
26 2022-11-05 13:43:48.878 INFO 10704 --- [          main]
    mao.j2cache_demo.J2cacheDemoApplication : Started J2cacheDemoApplication in
    2.561 seconds (JVM running for 3.309)
```

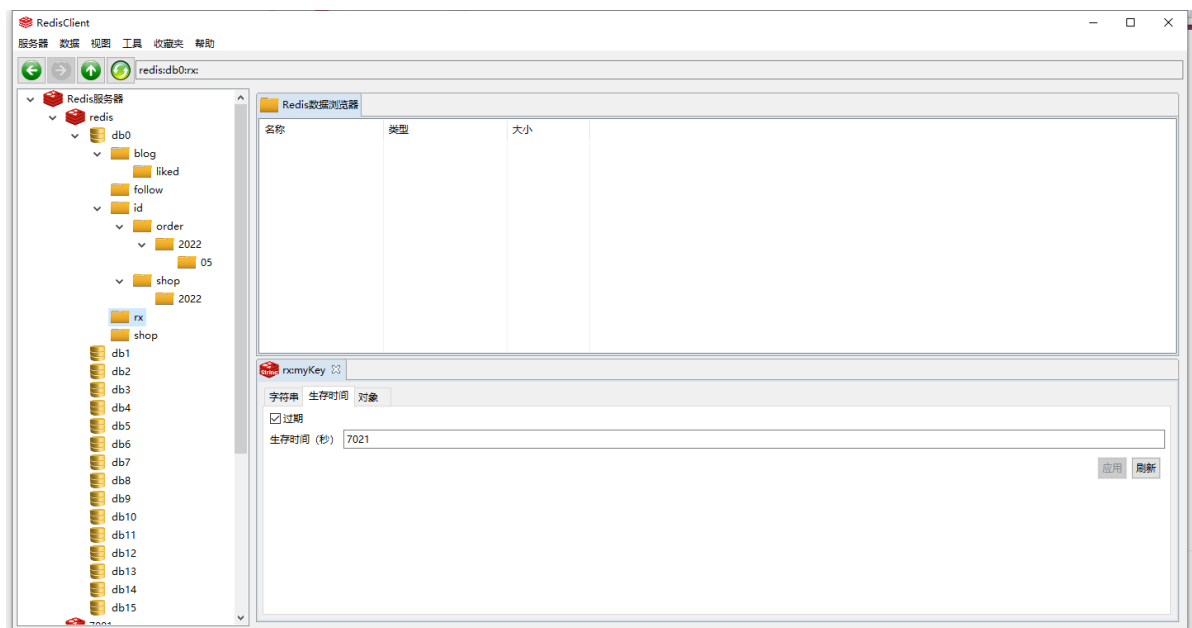
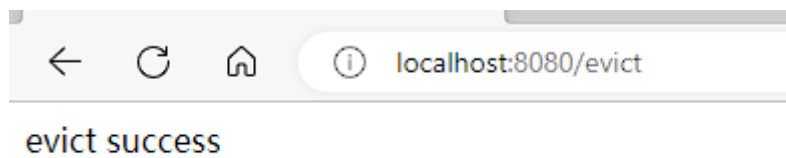
第八步：访问

<http://localhost:8080/getInfos>



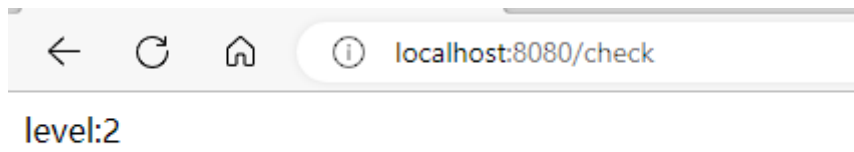


<http://localhost:8080/evict>

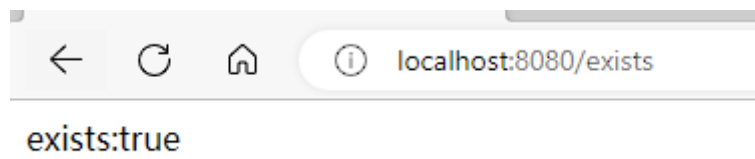


<http://localhost:8080/getInfos>

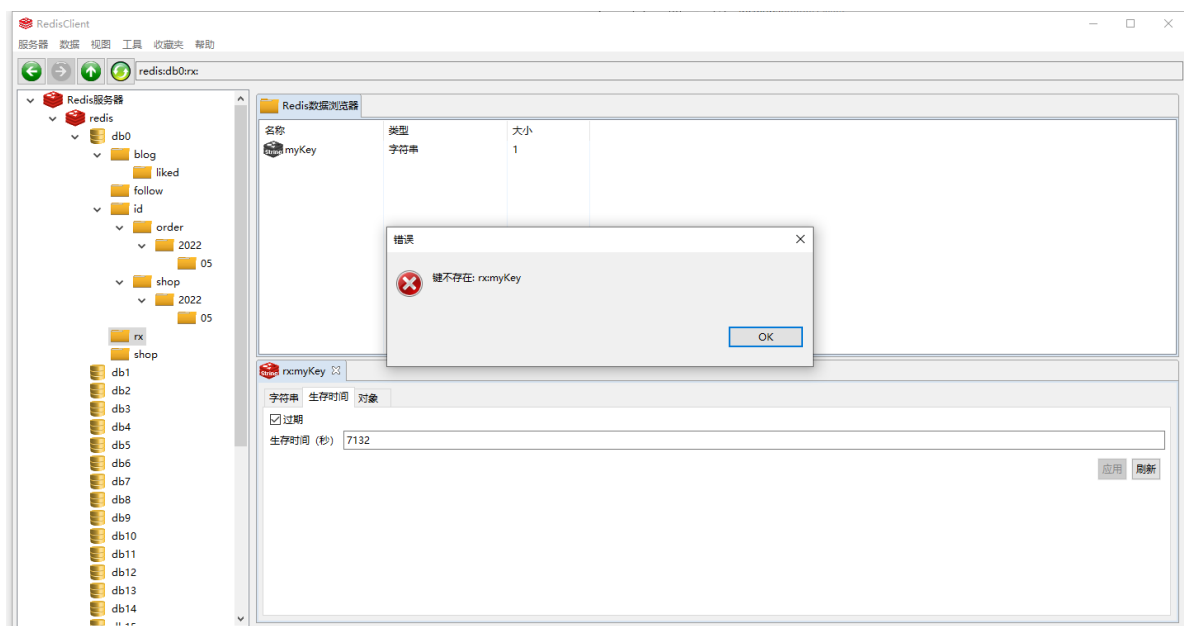
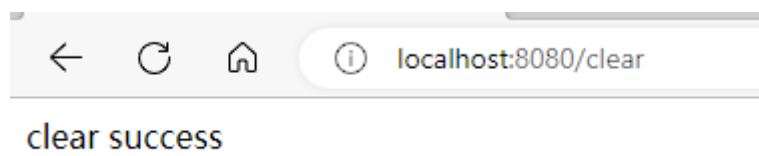
<http://localhost:8080/check>



<http://localhost:8080/exists>



<http://localhost:8080/clear>



无法启动解决

启动报以下错误:

```
1 Error starting ApplicationContext. To display the conditions report re-run
  your application with 'debug' enabled.
2 2022-11-05 13:32:38.028 ERROR 8520 --- [           main]
  o.s.boot.SpringApplication : Application run failed
3
4 org.springframework.beans.factory.UnsatisfiedDependencyException: Error
  creating bean with name 'testController': Unsatisfied dependency expressed
  through field 'cacheChannel'; nested exception is
  org.springframework.beans.factory.BeanCreationException: Error creating bean
  with name 'cacheChannel' defined in class path resource
  [net.oschina.j2cache.autoconfigure.J2CacheAutoConfiguration.class]: Bean
  instantiation via factory method failed; nested exception is
  org.springframework.beans.BeanInstantiationException: Failed to instantiate
  [net.oschina.j2cache.CacheChannel]: Factory method 'cacheChannel' threw
  exception; nested exception is
  java.lang.reflect.InaccessibleObjectException: Unable to make field private
  final java.math.BigInteger java.math.BigDecimal.intVal accessible: module
  java.base does not "opens java.math" to unnamed module @76908cc0
5   at
  org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProc
  essor$AutowiredFieldElement.resolveFieldValue(AutowiredAnnotationBeanPostPro
  cessor.java:659) ~[spring-beans-5.3.21.jar:5.3.21]
```

```
6      at
org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProc
essor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java
:639) ~[spring-beans-5.3.21.jar:5.3.21]
7      at
org.springframework.beans.factory.annotation.InjectionMetadata.inject(Inject
ionMetadata.java:119) ~[spring-beans-5.3.21.jar:5.3.21]
8      at
org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProc
essor.postProcessProperties(AutowiredAnnotationBeanPostProcessor.java:399) ~
[spring-beans-5.3.21.jar:5.3.21]
9      at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
.populateBean(AbstractAutowireCapableBeanFactory.java:1431) ~[spring-beans-
5.3.21.jar:5.3.21]
10     at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
.doCreateBean(AbstractAutowireCapableBeanFactory.java:619) ~[spring-beans-
5.3.21.jar:5.3.21]
11     at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
.createBean(AbstractAutowireCapableBeanFactory.java:542) ~[spring-beans-
5.3.21.jar:5.3.21]
12     at
org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBe
an$0(AbstractBeanFactory.java:335) ~[spring-beans-5.3.21.jar:5.3.21]
13     at
org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSi
ngleton(DefaultSingletonBeanRegistry.java:234) ~[spring-beans-
5.3.21.jar:5.3.21]
14     at
org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(Abst
ractBeanFactory.java:333) ~[spring-beans-5.3.21.jar:5.3.21]
15     at
org.springframework.beans.factory.support.AbstractBeanFactory.getBean(Abstra
ctBeanFactory.java:208) ~[spring-beans-5.3.21.jar:5.3.21]
16     at
org.springframework.beans.factory.support.DefaultListableBeanFactory.preInst
antiateSingletons(DefaultListableBeanFactory.java:955) ~[spring-beans-
5.3.21.jar:5.3.21]
17     at
org.springframework.context.support.AbstractApplicationContext.finishBeanFac
toryInitialization(AbstractApplicationContext.java:918) ~[spring-context-
5.3.21.jar:5.3.21]
18     at
org.springframework.context.support.AbstractApplicationContext.refresh(Abst
ractApplicationContext.java:583) ~[spring-context-5.3.21.jar:5.3.21]
19     at
org.springframework.boot.web.servlet.context.ServletWebServerApplicationCont
ext.refresh(ServletWebServerApplicationContext.java:147) ~[spring-boot-
2.7.1.jar:2.7.1]
20     at
org.springframework.boot.SpringApplication.refresh(SpringApplication.java:73
4) ~[spring-boot-2.7.1.jar:2.7.1]
21     at
org.springframework.boot.SpringApplication.refreshContext(SpringApplication.
java:408) ~[spring-boot-2.7.1.jar:2.7.1]
```

```

22      at
org.springframework.boot.SpringApplication.run(SpringApplication.java:308) ~
[spring-boot-2.7.1.jar:2.7.1]
23      at
org.springframework.boot.SpringApplication.run(SpringApplication.java:1306)
~[spring-boot-2.7.1.jar:2.7.1]
24      at
org.springframework.boot.SpringApplication.run(SpringApplication.java:1295)
~[spring-boot-2.7.1.jar:2.7.1]
25      at
mao.j2cache_demo.J2cacheDemoApplication.main(J2cacheDemoApplication.java:12)
~[classes/:na]
26  Caused by: org.springframework.beans.factory.BeanCreationException: Error
creating bean with name 'cacheChannel' defined in class path resource
[net.oschina.j2cache/autoconfigure/J2CacheAutoConfiguration.class]: Bean
instantiation via factory method failed; nested exception is
org.springframework.beans.BeanInstantiationException: Failed to instantiate
[net.oschina.j2cache.CacheChannel]: Factory method 'cacheChannel' threw
exception; nested exception is
java.lang.reflect.InaccessibleObjectException: Unable to make field private
final java.math.BigInteger java.math.BigDecimal.intVal accessible: module
java.base does not "opens java.math" to unnamed module @76908cc0
27      at
org.springframework.beans.factory.support.ConstructorResolver.instantiate(Con
structorResolver.java:658) ~[spring-beans-5.3.21.jar:5.3.21]
28      at
org.springframework.beans.factory.support.ConstructorResolver.instantiateUsi
ngFactoryMethod(ConstructorResolver.java:638) ~[spring-beans-
5.3.21.jar:5.3.21]
29      at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
.instantiateUsingFactoryMethod(AbstractAutowireCapableBeanFactory.java:1352)
~[spring-beans-5.3.21.jar:5.3.21]
30      at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
.createBeanInstance(AbstractAutowireCapableBeanFactory.java:1195) ~[spring-
beans-5.3.21.jar:5.3.21]
31      at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
.doCreateBean(AbstractAutowireCapableBeanFactory.java:582) ~[spring-beans-
5.3.21.jar:5.3.21]
32      at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
.createBean(AbstractAutowireCapableBeanFactory.java:542) ~[spring-beans-
5.3.21.jar:5.3.21]
33      at
org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBe
an$0(AbstractBeanFactory.java:335) ~[spring-beans-5.3.21.jar:5.3.21]
34      at
org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSi
ngleton(DefaultSingletonBeanRegistry.java:234) ~[spring-beans-
5.3.21.jar:5.3.21]
35      at
org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(Abst
ractBeanFactory.java:333) ~[spring-beans-5.3.21.jar:5.3.21]
36      at
org.springframework.beans.factory.support.AbstractBeanFactory.getBean(Abstra
ctBeanFactory.java:208) ~[spring-beans-5.3.21.jar:5.3.21]

```

```

37      at
org.springframework.beans.factory.config.DependencyDescriptor.resolveCandidate(DependencyDescriptor.java:276) ~[spring-beans-5.3.21.jar:5.3.21]
38      at
org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:1391) ~[spring-beans-5.3.21.jar:5.3.21]
39      at
org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:1311) ~[spring-beans-5.3.21.jar:5.3.21]
40      at
org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.resolveFieldValue(AutowiredAnnotationBeanPostProcessor.java:656) ~[spring-beans-5.3.21.jar:5.3.21]
41      ... 20 common frames omitted
42  Caused by: org.springframework.beans.BeanInstantiationException: Failed to instantiate [net.oschina.j2cache.CacheChannel]: Factory method
'cacheChannel' threw exception; nested exception is
java.lang.reflect.InaccessibleObjectException: Unable to make field private final java.math.BigInteger java.math.BigDecimal.intVal accessible: module
java.base does not "opens java.math" to unnamed module @76908cc0
43      at
org.springframework.beans.factory.support.SimpleInstantiationStrategy.instantiate(SimpleInstantiationStrategy.java:185) ~[spring-beans-5.3.21.jar:5.3.21]
44      at
org.springframework.beans.factory.support.ConstructorResolver.instantiate(ConstructorResolver.java:653) ~[spring-beans-5.3.21.jar:5.3.21]
45      ... 33 common frames omitted
46  Caused by: java.lang.reflect.InaccessibleObjectException: Unable to make
field private final java.math.BigInteger java.math.BigDecimal.intVal
accessible: module java.base does not "opens java.math" to unnamed module
@76908cc0
47      at
java.base/java.lang.reflect.AccessibleObject.checkCanSetAccessible(AccessibleObject.java:357) ~[na:na]
48      at
java.base/java.lang.reflect.AccessibleObject.checkCanSetAccessible(AccessibleObject.java:297) ~[na:na]
49      at
java.base/java.lang.reflect.Field.checkCanSetAccessible(Field.java:177) ~[na:na]
50      at java.base/java.lang.reflect.Field.setAccessible(Field.java:171) ~[na:na]
51      at
org.nustaq.serialization.FSTClazzInfo.createFieldInfo(FSTClazzInfo.java:512) ~[fst-2.57.jar:na]
52      at
org.nustaq.serialization.FSTClazzInfo.createFields(FSTClazzInfo.java:368) ~[fst-2.57.jar:na]
53      at org.nustaq.serialization.FSTClazzInfo.<init>(FSTClazzInfo.java:129) ~[fst-2.57.jar:na]
54      at
org.nustaq.serialization.FSTClazzInfoRegistry.getCLInfo(FSTClazzInfoRegistry.java:129) ~[fst-2.57.jar:na]

```



```

55      at
org.nustaq.serialization.FSTClazzNameRegistry.addClassMapping(FSTClazzNameRe
gistry.java:98) ~[fst-2.57.jar:na]
56      at
org.nustaq.serialization.FSTClazzNameRegistry.registerClassNoLookup(FSTClazz
NameRegistry.java:85) ~[fst-2.57.jar:na]
57      at
org.nustaq.serialization.FSTClazzNameRegistry.registerClass(FSTClazzNameRegi
stry.java:81) ~[fst-2.57.jar:na]
58      at
org.nustaq.serialization.FSTConfiguration.addDefaultClazzes(FSTConfiguration
.java:814) ~[fst-2.57.jar:na]
59      at
org.nustaq.serialization.FSTConfiguration.initDefaultFstConfigurationInterna
l(FSTConfiguration.java:477) ~[fst-2.57.jar:na]
60      at
org.nustaq.serialization.FSTConfiguration.createDefaultConfiguration(FSTConf
iguration.java:472) ~[fst-2.57.jar:na]
61      at
org.nustaq.serialization.FSTConfiguration.createDefaultConfiguration(FSTConf
iguration.java:464) ~[fst-2.57.jar:na]
62      at
org.nustaq.serialization.FSTConfiguration.getDefaultConfiguration(FSTConfigu
ration.java:204) ~[fst-2.57.jar:na]
63      at net.oschina.j2cache.util.FSTSerializer.<init>(FSTSerializer.java:30)
~[j2cache-core-2.8.0-release.jar:na]
64      at
net.oschina.j2cache.util.SerializationUtils.init(SerializationUtils.java:47)
~[j2cache-core-2.8.0-release.jar:na]
65      at
net.oschina.j2cache.J2CacheBuilder.initFromConfig(J2CacheBuilder.java:108) ~
[j2cache-core-2.8.0-release.jar:na]
66      at net.oschina.j2cache.J2CacheBuilder.getChannel(J2CacheBuilder.java:65)
~[j2cache-core-2.8.0-release.jar:na]
67      at
net.oschina.j2cache.autoconfigure.J2CacheAutoConfiguration.cacheChannel(J2Ca
cheAutoConfiguration.java:43) ~[j2cache-spring-boot2-starter-2.8.0-
release.jar:na]
68      at
net.oschina.j2cache.autoconfigure.J2CacheAutoConfiguration$$EnhancerBySpring
CGLIB$$81aadd46.cacheChannel$0(<generated>) ~[j2cache-spring-boot2-
starter-2.8.0-release.jar:na]
69      at
net.oschina.j2cache.autoconfigure.J2CacheAutoConfiguration$$EnhancerBySpring
CGLIB$$81aadd46$$FastClassBySpringCGLIB$$7ae942f0.invoke(<generated>) ~
[j2cache-spring-boot2-starter-2.8.0-release.jar:na]
70      at
org.springframework.cglib.proxy.MethodProxy.invokeSuper(MethodProxy.java:244
) ~[spring-core-5.3.21.jar:5.3.21]
71      at
org.springframework.context.annotation.ConfigurationClassEnhancer$BeanMethod
Interceptor.intercept(ConfigurationClassEnhancer.java:331) ~[spring-context-
5.3.21.jar:5.3.21]
72      at
net.oschina.j2cache.autoconfigure.J2CacheAutoConfiguration$$EnhancerBySpring
CGLIB$$81aadd46.cacheChannel(<generated>) ~[j2cache-spring-boot2-starter-
2.8.0-release.jar:na]

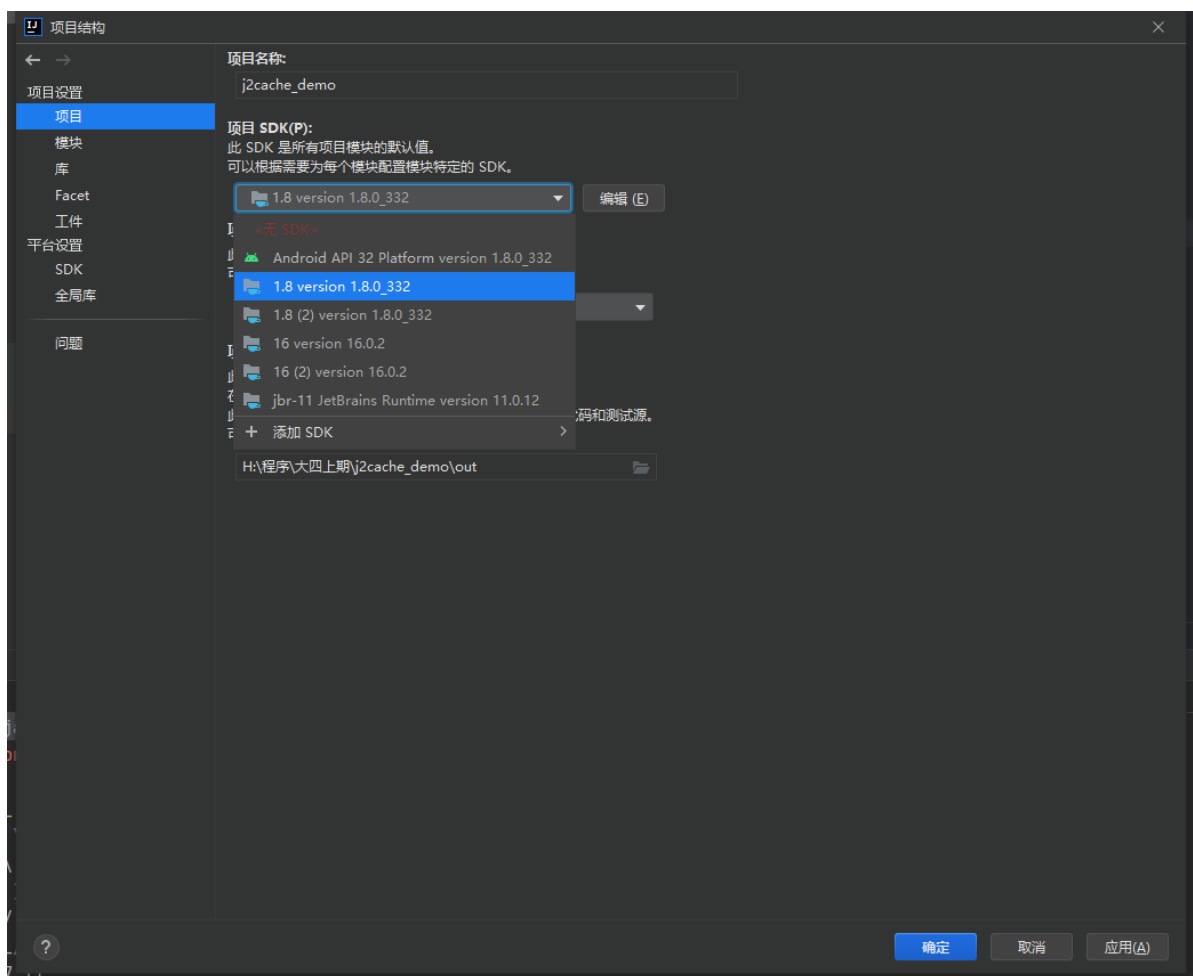
```

```

73     at
    java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native
    Method) ~[na:na]
74     at
    java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodA
    ccessorImpl.java:78) ~[na:na]
75     at
    java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(Delegatin
    gMethodAccessorImpl.java:43) ~[na:na]
76     at java.base/java.lang.reflect.Method.invoke(Method.java:567) ~[na:na]
77     at
    org.springframework.beans.factory.support.SimpleInstantiationStrategy.instan
    tiate(SimpleInstantiationStrategy.java:154) ~[spring-beans-
    5.3.21.jar:5.3.21]
78     ... 34 common frames omitted
79

```

解决方案：将jdk版本更改为1.8



测试缓存击穿

第一步：修改TestController

```
1 package mao.j2cache_demo.controller;
2
3 import net.oschina.j2cache.CacheChannel;
4 import net.oschina.j2cache.CacheObject;
5 import org.slf4j.Logger;
6 import org.slf4j.LoggerFactory;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.RestController;
10
11 import java.util.ArrayList;
12 import java.util.List;
13
14 /**
15  * Project name(项目名称): j2cache_demo
16  * Package(包名): mao.j2cache_demo.controller
17  * Class(类名): TestController
18  * Author(作者): mao
19  * Author QQ: 1296193245
20  * GitHub: https://github.com/maomao124/
21  * Date(创建日期): 2022/11/5
22  * Time(创建时间): 13:22
23  * Version(版本): 1.0
24  * Description(描述): 无
25  */
26
27 @RestController
28 public class TestController
29 {
30
31     private static final Logger log =
32         LoggerFactory.getLogger(TestController.class);
33
34     @Autowired
35     private CacheChannel cacheChannel;
36
37     private final String key = "myKey";
38     private final String region = "rx";
39
40     @GetMapping("/getInfos")
41     public List<String> getInfos()
42     {
43         CacheObject cacheObject = cacheChannel.get(region, key);
44         if (cacheObject.getValue() == null)
```

```

45     {
46         log.info("查询数据库");
47         //缓存中没有找到, 查询数据库获得
48         List<String> data = new ArrayList<>();
49         data.add("info1");
50         data.add("info2");
51         try
52         {
53             Thread.sleep(9);
54         }
55         catch (InterruptedException e)
56         {
57             e.printStackTrace();
58         }
59         //放入缓存
60         cacheChannel.set(region, key, data);
61         return data;
62     }
63     return (List<String>) cacheObject.getValue();
64 }
65
66 /**
67  * 清理指定缓存
68  *
69  * @return {@link String}
70  */
71 @GetMapping("/evict")
72 public String evict()
73 {
74     cacheChannel.evict(region, key);
75     return "evict success";
76 }
77
78 /**
79  * 检测存在哪级缓存
80  *
81  * @return {@link String}
82  */
83 @GetMapping("/check")
84 public String check()
85 {
86     int check = cacheChannel.check(region, key);
87     return "level:" + check;
88 }
89
90 /**
91  * 检测缓存数据是否存在
92  *
93  * @return {@link String}
94  */
95 @GetMapping("/exists")
96 public String exists()
97 {
98     boolean exists = cacheChannel.exists(region, key);
99     return "exists:" + exists;
100 }
101
102 /**

```

```
103      * 清理指定区域的缓存
104      *
105      * @return {@link String}
106      */
107      @GetMapping("/clear")
108      public String clear()
109      {
110          cacheChannel.clear(region);
111          return "clear success";
112      }
113  }
```

第二步：启动程序，并打开jmeter

setUp线程组

名称:

注释:

在取样器错误后要执行的动作

☒ 继续 ☐ 启动下一进程循环 ☐ 停止线程 ☐ 停止测试 ☐ 立即停止测试

线程属性

线程数:

Ramp-Up时间(秒):

循环次数 ☒ 永远

☒ Same user on each iteration

☐ 调度器

持续时间(秒)

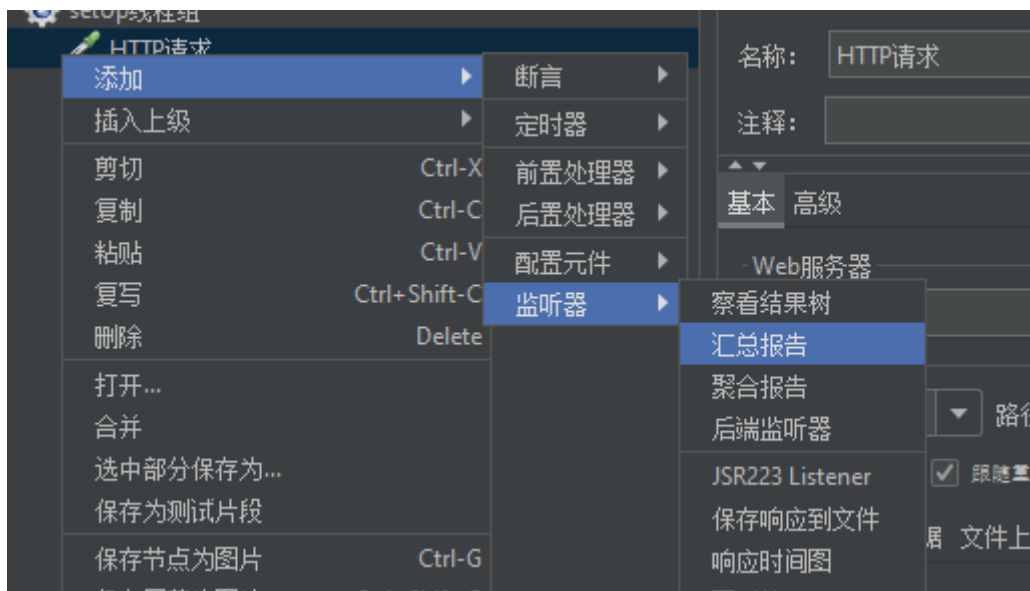
启动延迟(秒)

300线程并发

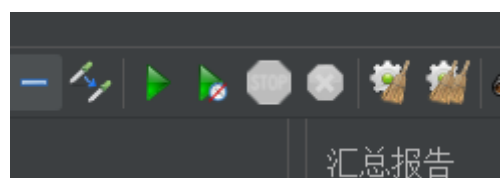
第三步：设置http请求



第四步：添加监听器

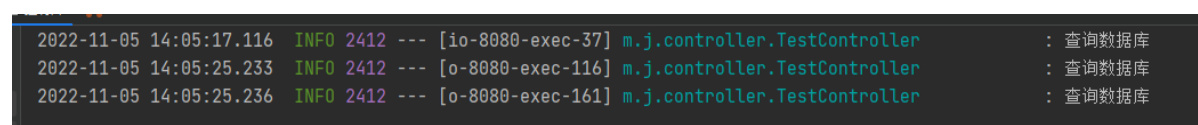
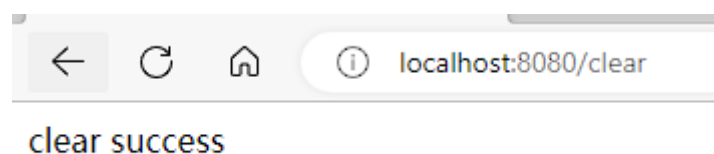


第五步：启动jmeter



第六步：主动清除缓存

<http://localhost:8080/clear>



```

2022-11-05 14:05:17.116 INFO 2412 --- [io-8080-exec-37] m.j.controller.TestController : 查询数据库
2022-11-05 14:05:25.233 INFO 2412 --- [o-8080-exec-116] m.j.controller.TestController : 查询数据库
2022-11-05 14:05:25.236 INFO 2412 --- [o-8080-exec-161] m.j.controller.TestController : 查询数据库
2022-11-05 14:05:50.401 INFO 2412 --- [o-8080-exec-112] m.j.controller.TestController : 查询数据库
2022-11-05 14:05:50.413 INFO 2412 --- [o-8080-exec-117] m.j.controller.TestController : 查询数据库
2022-11-05 14:05:50.405 INFO 2412 --- [o-8080-exec-160] m.j.controller.TestController : 查询数据库

```

汇总报告										
名称: 汇总报告										
注释:										
所有数据写入一个文件										
文件名							浏览	显示日志内容:	<input type="checkbox"/> 仅错误日志 <input type="checkbox"/> 仅成功日志	配置
Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	3927436	12	0	38680	129.74	0.82%	23508.2/sec	4762.04	2823.48	207.4
总体	3927448	12	0	38680	129.74	0.82%	23508.1/sec	4762.03	2823.48	207.4

有时候能同时通过两个请求，有时候能同时通过3个请求，没有完全解决缓存击穿问题，但是影响不大

拿自己实现的缓存做比较

```

1 package mao.j2cache_demo.controller;
2
3 import net.oschina.j2cache.CacheChannel;
4 import net.oschina.j2cache.CacheObject;
5 import org.slf4j.Logger;
6 import org.slf4j.LoggerFactory;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.RestController;
10
11 import java.util.ArrayList;
12 import java.util.List;
13
14 /**
15  * Project name(项目名称): j2cache_demo
16  * Package(包名): mao.j2cache_demo.controller
17  * Class(类名): TestController
18  * Author(作者): mao
19  * Author QQ: 1296193245
20  * GitHub: https://github.com/maomao124/
21  * Date(创建日期): 2022/11/5
22  * Time(创建时间): 13:22
23  * Version(版本): 1.0
24  * Description(描述): 无
25  */
26
27 @RestController
28 public class TestController

```



```

29 {
30
31     private static final Logger log =
LoggerFactory.getLogger(TestController.class);
32
33     @Autowired
34     private CacheChannel cacheChannel;
35
36     private final String key = "myKey";
37     private final String region = "rx";
38
39
40     @GetMapping("/getInfos")
41     public List<String> getInfos()
42     {
43         CacheObject cacheObject = cacheChannel.get(region, key);
44         if (cacheObject.getValue() == null)
45         {
46             log.info("查询数据库");
47             //缓存中没有找到，查询数据库获得
48             List<String> data = new ArrayList<>();
49             data.add("info1");
50             data.add("info2");
51             //放入缓存
52             cacheChannel.set(region, key, data);
53             return data;
54         }
55         return (List<String>) cacheObject.getValue();
56     }
57
58     private String cache = null;
59
60     @GetMapping("/getInfos2")
61     public String getInfos2()
62     {
63         if (cache == null)
64         {
65             log.info("查询数据库2");
66             try
67             {
68                 Thread.sleep(10);
69             }
70             catch (InterruptedException e)
71             {
72                 e.printStackTrace();
73             }
74             cache = "hello";
75         }
76         else
77         {
78             return cache;
79         }
80         return cache;
81     }
82
83     /**
84     * 清理指定缓存
85     *

```

```

86     * @return {@link String}
87     */
88     @GetMapping("/evict")
89     public String evict()
90     {
91         cacheChannel.evict(region, key);
92         return "evict success";
93     }
94
95     /**
96      * 检测存在哪级缓存
97      *
98      * @return {@link String}
99      */
100    @GetMapping("/check")
101    public String check()
102    {
103        int check = cacheChannel.check(region, key);
104        return "level:" + check;
105    }
106
107    /**
108     * 检测缓存数据是否存在
109     *
110     * @return {@link String}
111     */
112    @GetMapping("/exists")
113    public String exists()
114    {
115        boolean exists = cacheChannel.exists(region, key);
116        return "exists:" + exists;
117    }
118
119    /**
120     * 清理指定区域的缓存
121     *
122     * @return {@link String}
123     */
124    @GetMapping("/clear")
125    public String clear()
126    {
127        cache = null;
128        cacheChannel.clear(region);
129        return "clear success";
130    }
131 }

```

HTTP请求

名称: HTTP请求

注释:

基本高级

Web服务器

协议: 服务器名称或IP:

HTTP请求

GET 路径: http://localhost:8080/getInfos2

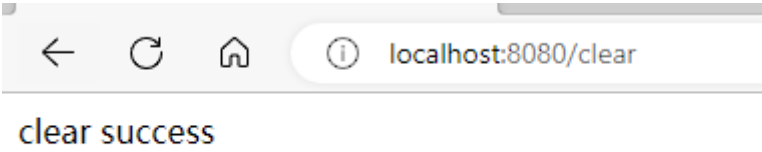
☐ 自动重定向 ☒ 跟随重定向 ☒ 使用 KeepAlive ☐ 对POST使用multipart / form-data ☐ 与浏览器兼容

参数消息体数据文件上传

同请

名称:

值



```
1 2022-11-05 14:20:07.152 INFO 4668 --- [o-8080-exec-100]
   m.j.controller.TestController      : 查询数据库2
2 2022-11-05 14:20:07.152 INFO 4668 --- [o-8080-exec-103]
   m.j.controller.TestController      : 查询数据库2
3 2022-11-05 14:20:07.152 INFO 4668 --- [io-8080-exec-69]
   m.j.controller.TestController      : 查询数据库2
4 2022-11-05 14:20:07.152 INFO 4668 --- [o-8080-exec-116]
   m.j.controller.TestController      : 查询数据库2
5 2022-11-05 14:20:07.152 INFO 4668 --- [o-8080-exec-198]
   m.j.controller.TestController      : 查询数据库2
6 2022-11-05 14:20:07.152 INFO 4668 --- [o-8080-exec-131]
   m.j.controller.TestController      : 查询数据库2
7 2022-11-05 14:20:07.152 INFO 4668 --- [io-8080-exec-40]
   m.j.controller.TestController      : 查询数据库2
8 2022-11-05 14:20:07.152 INFO 4668 --- [o-8080-exec-157]
   m.j.controller.TestController      : 查询数据库2
9 2022-11-05 14:20:07.152 INFO 4668 --- [o-8080-exec-150]
   m.j.controller.TestController      : 查询数据库2
10 2022-11-05 14:20:07.152 INFO 4668 --- [io-8080-exec-24]
   m.j.controller.TestController      : 查询数据库2
11 2022-11-05 14:20:07.152 INFO 4668 --- [io-8080-exec-53]
   m.j.controller.TestController      : 查询数据库2
```

[illegible]

```

41 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-85]
    m.j.controller.TestController      : 查询数据库2
42 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-184]
    m.j.controller.TestController      : 查询数据库2
43 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-97]
    m.j.controller.TestController      : 查询数据库2
44 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-93]
    m.j.controller.TestController      : 查询数据库2
45 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-89]
    m.j.controller.TestController      : 查询数据库2
46 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-51]
    m.j.controller.TestController      : 查询数据库2
47 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-38]
    m.j.controller.TestController      : 查询数据库2
48 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-194]
    m.j.controller.TestController      : 查询数据库2
49 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-177]
    m.j.controller.TestController      : 查询数据库2
50 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-197]
    m.j.controller.TestController      : 查询数据库2
51 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-33]
    m.j.controller.TestController      : 查询数据库2
52 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-120]
    m.j.controller.TestController      : 查询数据库2
53 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-160]
    m.j.controller.TestController      : 查询数据库2
54 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-108]
    m.j.controller.TestController      : 查询数据库2
55 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-17]
    m.j.controller.TestController      : 查询数据库2
56 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-96]
    m.j.controller.TestController      : 查询数据库2
57 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-12]
    m.j.controller.TestController      : 查询数据库2
58 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-22]
    m.j.controller.TestController      : 查询数据库2
59 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-98]
    m.j.controller.TestController      : 查询数据库2
60 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-72]
    m.j.controller.TestController      : 查询数据库2
61 2022-11-05 14:20:07.155 INFO 4668 --- [nio-8080-exec-4]
    m.j.controller.TestController      : 查询数据库2
62 2022-11-05 14:20:07.155 INFO 4668 --- [io-8080-exec-28]
    m.j.controller.TestController      : 查询数据库2
63 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-171]
    m.j.controller.TestController      : 查询数据库2
64 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-91]
    m.j.controller.TestController      : 查询数据库2
65 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-151]
    m.j.controller.TestController      : 查询数据库2
66 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-34]
    m.j.controller.TestController      : 查询数据库2
67 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-110]
    m.j.controller.TestController      : 查询数据库2
68 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-154]
    m.j.controller.TestController      : 查询数据库2
69 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-174]
    m.j.controller.TestController      : 查询数据库2

```

```

70 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-64]
    m.j.controller.TestController      : 查询数据库2
71 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-186]
    m.j.controller.TestController      : 查询数据库2
72 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-161]
    m.j.controller.TestController      : 查询数据库2
73 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-66]
    m.j.controller.TestController      : 查询数据库2
74 2022-11-05 14:20:07.155 INFO 4668 --- [io-8080-exec-73]
    m.j.controller.TestController      : 查询数据库2
75 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-54]
    m.j.controller.TestController      : 查询数据库2
76 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-193]
    m.j.controller.TestController      : 查询数据库2
77 2022-11-05 14:20:07.155 INFO 4668 --- [io-8080-exec-70]
    m.j.controller.TestController      : 查询数据库2
78 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-39]
    m.j.controller.TestController      : 查询数据库2
79 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-57]
    m.j.controller.TestController      : 查询数据库2
80 2022-11-05 14:20:07.153 INFO 4668 --- [io-8080-exec-42]
    m.j.controller.TestController      : 查询数据库2
81 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-79]
    m.j.controller.TestController      : 查询数据库2
82 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-56]
    m.j.controller.TestController      : 查询数据库2
83 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-47]
    m.j.controller.TestController      : 查询数据库2
84 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-67]
    m.j.controller.TestController      : 查询数据库2
85 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-44]
    m.j.controller.TestController      : 查询数据库2
86 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-80]
    m.j.controller.TestController      : 查询数据库2
87 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-191]
    m.j.controller.TestController      : 查询数据库2
88 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-122]
    m.j.controller.TestController      : 查询数据库2
89 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-130]
    m.j.controller.TestController      : 查询数据库2
90 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-86]
    m.j.controller.TestController      : 查询数据库2
91 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-166]
    m.j.controller.TestController      : 查询数据库2
92 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-136]
    m.j.controller.TestController      : 查询数据库2
93 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-169]
    m.j.controller.TestController      : 查询数据库2
94 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-125]
    m.j.controller.TestController      : 查询数据库2
95 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-137]
    m.j.controller.TestController      : 查询数据库2
96 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-129]
    m.j.controller.TestController      : 查询数据库2
97 2022-11-05 14:20:07.156 INFO 4668 --- [nio-8080-exec-7]
    m.j.controller.TestController      : 查询数据库2
98 2022-11-05 14:20:07.154 INFO 4668 --- [nio-8080-exec-1]
    m.j.controller.TestController      : 查询数据库2

```

```

99 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-43]
    m.j.controller.TestController      : 查询数据库2
100 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-139]
    m.j.controller.TestController      : 查询数据库2
101 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-78]
    m.j.controller.TestController      : 查询数据库2
102 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-118]
    m.j.controller.TestController      : 查询数据库2
103 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-172]
    m.j.controller.TestController      : 查询数据库2
104 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-145]
    m.j.controller.TestController      : 查询数据库2
105 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-68]
    m.j.controller.TestController      : 查询数据库2
106 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-74]
    m.j.controller.TestController      : 查询数据库2
107 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-162]
    m.j.controller.TestController      : 查询数据库2
108 2022-11-05 14:20:07.153 INFO 4668 --- [o-8080-exec-152]
    m.j.controller.TestController      : 查询数据库2
109 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-55]
    m.j.controller.TestController      : 查询数据库2
110 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-48]
    m.j.controller.TestController      : 查询数据库2
111 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-149]
    m.j.controller.TestController      : 查询数据库2
112 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-115]
    m.j.controller.TestController      : 查询数据库2
113 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-30]
    m.j.controller.TestController      : 查询数据库2
114 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-188]
    m.j.controller.TestController      : 查询数据库2
115 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-146]
    m.j.controller.TestController      : 查询数据库2
116 2022-11-05 14:20:07.154 INFO 4668 --- [io-8080-exec-14]
    m.j.controller.TestController      : 查询数据库2
117 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-143]
    m.j.controller.TestController      : 查询数据库2
118 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-180]
    m.j.controller.TestController      : 查询数据库2
119 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-132]
    m.j.controller.TestController      : 查询数据库2
120 2022-11-05 14:20:07.157 INFO 4668 --- [nio-8080-exec-8]
    m.j.controller.TestController      : 查询数据库2
121 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-58]
    m.j.controller.TestController      : 查询数据库2
122 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-142]
    m.j.controller.TestController      : 查询数据库2
123 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-92]
    m.j.controller.TestController      : 查询数据库2
124 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-165]
    m.j.controller.TestController      : 查询数据库2
125 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-87]
    m.j.controller.TestController      : 查询数据库2
126 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-173]
    m.j.controller.TestController      : 查询数据库2
127 2022-11-05 14:20:07.154 INFO 4668 --- [o-8080-exec-114]
    m.j.controller.TestController      : 查询数据库2

```



```

128 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-164]
    m.j.controller.TestController      : 查询数据库2
129 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-156]
    m.j.controller.TestController      : 查询数据库2
130 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-126]
    m.j.controller.TestController      : 查询数据库2
131 2022-11-05 14:20:07.155 INFO 4668 --- [nio-8080-exec-9]
    m.j.controller.TestController      : 查询数据库2
132 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-148]
    m.j.controller.TestController      : 查询数据库2
133 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-104]
    m.j.controller.TestController      : 查询数据库2
134 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-15]
    m.j.controller.TestController      : 查询数据库2
135 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-182]
    m.j.controller.TestController      : 查询数据库2
136 2022-11-05 14:20:07.155 INFO 4668 --- [nio-8080-exec-3]
    m.j.controller.TestController      : 查询数据库2
137 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-21]
    m.j.controller.TestController      : 查询数据库2
138 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-147]
    m.j.controller.TestController      : 查询数据库2
139 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-102]
    m.j.controller.TestController      : 查询数据库2
140 2022-11-05 14:20:07.155 INFO 4668 --- [o-8080-exec-155]
    m.j.controller.TestController      : 查询数据库2
141 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-128]
    m.j.controller.TestController      : 查询数据库2
142 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-106]
    m.j.controller.TestController      : 查询数据库2
143 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-199]
    m.j.controller.TestController      : 查询数据库2
144 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-107]
    m.j.controller.TestController      : 查询数据库2
145 2022-11-05 14:20:07.156 INFO 4668 --- [io-8080-exec-32]
    m.j.controller.TestController      : 查询数据库2
146 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-11]
    m.j.controller.TestController      : 查询数据库2
147 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-196]
    m.j.controller.TestController      : 查询数据库2
148 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-153]
    m.j.controller.TestController      : 查询数据库2
149 2022-11-05 14:20:07.156 INFO 4668 --- [io-8080-exec-82]
    m.j.controller.TestController      : 查询数据库2
150 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-36]
    m.j.controller.TestController      : 查询数据库2
151 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-178]
    m.j.controller.TestController      : 查询数据库2
152 2022-11-05 14:20:07.156 INFO 4668 --- [io-8080-exec-16]
    m.j.controller.TestController      : 查询数据库2
153 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-18]
    m.j.controller.TestController      : 查询数据库2
154 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-109]
    m.j.controller.TestController      : 查询数据库2
155 2022-11-05 14:20:07.156 INFO 4668 --- [io-8080-exec-65]
    m.j.controller.TestController      : 查询数据库2
156 2022-11-05 14:20:07.156 INFO 4668 --- [o-8080-exec-141]
    m.j.controller.TestController      : 查询数据库2

```



```

157 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-189]
    m.j.controller.TestController      : 查询数据库2
158 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-46]
    m.j.controller.TestController      : 查询数据库2
159 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-52]
    m.j.controller.TestController      : 查询数据库2
160 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-41]
    m.j.controller.TestController      : 查询数据库2
161 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-27]
    m.j.controller.TestController      : 查询数据库2
162 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-138]
    m.j.controller.TestController      : 查询数据库2
163 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-144]
    m.j.controller.TestController      : 查询数据库2
164 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-20]
    m.j.controller.TestController      : 查询数据库2
165 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-117]
    m.j.controller.TestController      : 查询数据库2
166 2022-11-05 14:20:07.157 INFO 4668 --- [o-8080-exec-134]
    m.j.controller.TestController      : 查询数据库2
167 2022-11-05 14:20:07.157 INFO 4668 --- [io-8080-exec-45]
    m.j.controller.TestController      : 查询数据库2
168 2022-11-05 14:20:07.158 INFO 4668 --- [nio-8080-exec-5]
    m.j.controller.TestController      : 查询数据库2
169 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-113]
    m.j.controller.TestController      : 查询数据库2
170 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-62]
    m.j.controller.TestController      : 查询数据库2
171 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-133]
    m.j.controller.TestController      : 查询数据库2
172 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-183]
    m.j.controller.TestController      : 查询数据库2
173 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-88]
    m.j.controller.TestController      : 查询数据库2
174 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-101]
    m.j.controller.TestController      : 查询数据库2
175 2022-11-05 14:20:07.158 INFO 4668 --- [nio-8080-exec-2]
    m.j.controller.TestController      : 查询数据库2
176 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-185]
    m.j.controller.TestController      : 查询数据库2
177 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-121]
    m.j.controller.TestController      : 查询数据库2
178 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-95]
    m.j.controller.TestController      : 查询数据库2
179 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-81]
    m.j.controller.TestController      : 查询数据库2
180 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-175]
    m.j.controller.TestController      : 查询数据库2
181 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-163]
    m.j.controller.TestController      : 查询数据库2
182 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-112]
    m.j.controller.TestController      : 查询数据库2
183 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-187]
    m.j.controller.TestController      : 查询数据库2
184 2022-11-05 14:20:07.158 INFO 4668 --- [o-8080-exec-168]
    m.j.controller.TestController      : 查询数据库2
185 2022-11-05 14:20:07.158 INFO 4668 --- [io-8080-exec-25]
    m.j.controller.TestController      : 查询数据库2

```

```
186 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-124]
    m.j.controller.TestController          : 查询数据库2
187 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-119]
    m.j.controller.TestController          : 查询数据库2
188 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-75]
    m.j.controller.TestController          : 查询数据库2
189 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-29]
    m.j.controller.TestController          : 查询数据库2
190 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-59]
    m.j.controller.TestController          : 查询数据库2
191 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-105]
    m.j.controller.TestController          : 查询数据库2
192 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-50]
    m.j.controller.TestController          : 查询数据库2
193 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-19]
    m.j.controller.TestController          : 查询数据库2
194 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-158]
    m.j.controller.TestController          : 查询数据库2
195 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-127]
    m.j.controller.TestController          : 查询数据库2
196 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-167]
    m.j.controller.TestController          : 查询数据库2
197 2022-11-05 14:20:07.159 INFO 4668 --- [io-8080-exec-77]
    m.j.controller.TestController          : 查询数据库2
198 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-181]
    m.j.controller.TestController          : 查询数据库2
199 2022-11-05 14:20:07.159 INFO 4668 --- [o-8080-exec-176]
    m.j.controller.TestController          : 查询数据库2
```

自己实现的缓存被查询了很多次，对数据库的影响大

测试缓存穿透

第一步：修改TestController

```
1 package mao.j2cache_demo.controller;
2
3 import net.oschina.j2cache.CacheChannel;
4 import net.oschina.j2cache.CacheObject;
5 import org.slf4j.Logger;
6 import org.slf4j.LoggerFactory;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.web.bind.annotation.GetMapping;
```

```

9  import org.springframework.web.bind.annotation.RestController;
10
11  import java.util.ArrayList;
12  import java.util.List;
13
14  /**
15   * Project name(项目名称): j2cache_demo
16   * Package(包名): mao.j2cache_demo.controller
17   * Class(类名): TestController
18   * Author(作者): mao
19   * Author QQ: 1296193245
20   * Github: https://github.com/maomao124/
21   * Date(创建日期): 2022/11/5
22   * Time(创建时间): 13:22
23   * Version(版本): 1.0
24   * Description(描述): 无
25   */
26
27  @RestController
28  public class TestController
29  {
30
31      private static final Logger log =
32      LoggerFactory.getLogger(TestController.class);
33
34      @Autowired
35      private CacheChannel cacheChannel;
36
37      private final String key = "myKey";
38      private final String region = "rx";
39
40      @GetMapping("/getInfos")
41      public List<String> getInfos()
42      {
43          CacheObject cacheObject = cacheChannel.get(region, key);
44          if (cacheObject.getValue() == null)
45          {
46              log.info("查询数据库");
47              //缓存中没有找到，查询数据库获得
48              List<String> data = new ArrayList<>();
49              data.add("info1");
50              data.add("info2");
51              try
52              {
53                  Thread.sleep(9);
54              }
55              catch (InterruptedException e)
56              {
57                  e.printStackTrace();
58              }
59              //放入缓存
60              cacheChannel.set(region, key, data);
61              return data;
62          }
63          return (List<String>) cacheObject.getValue();
64      }
65

```

```

66     private String cache = null;
67
68     @GetMapping("/getInfos2")
69     public String getInfos2()
70     {
71         if (cache == null)
72         {
73             log.info("查询数据库2");
74             try
75             {
76                 Thread.sleep(10);
77             }
78             catch (InterruptedException e)
79             {
80                 e.printStackTrace();
81             }
82             cache = "hello";
83         }
84         else
85         {
86             return cache;
87         }
88         return cache;
89     }
90
91
92     @GetMapping("/getInfos3")
93     public List<String> getInfos3()
94     {
95         CacheObject cacheObject = cacheChannel.get(region, key);
96         if (cacheObject.getValue() == null)
97         {
98             log.info("查询数据库3");
99             //缓存中没有找到，查询数据库获得
100             try
101             {
102                 Thread.sleep(9);
103             }
104             catch (InterruptedException e)
105             {
106                 e.printStackTrace();
107             }
108             //放入缓存
109             cacheChannel.set(region, key, null);
110             return null;
111         }
112         return null;
113     }
114
115     /**
116      * 清理指定缓存
117      *
118      * @return {@link String}
119      */
120     @GetMapping("/evict")
121     public String evict()
122     {
123         cacheChannel.evict(region, key);

```

```

124         return "evict success";
125     }
126
127     /**
128      * 检测存在哪级缓存
129      *
130      * @return {@link String}
131      */
132     @GetMapping("/check")
133     public String check()
134     {
135         int check = cacheChannel.check(region, key);
136         return "level:" + check;
137     }
138
139     /**
140      * 检测缓存数据是否存在
141      *
142      * @return {@link String}
143      */
144     @GetMapping("/exists")
145     public String exists()
146     {
147         boolean exists = cacheChannel.exists(region, key);
148         return "exists:" + exists;
149     }
150
151     /**
152      * 清理指定区域的缓存
153      *
154      * @return {@link String}
155      */
156     @GetMapping("/clear")
157     public String clear()
158     {
159         cache = null;
160         cacheChannel.clear(region);
161         return "clear success";
162     }
163 }

```

第二步：设置http请求

HTTP请求

名称：HTTP请求

注释：

基本 高级

Web服务器

协议：

服务器名称或IP：

HTTP请求

GET

路径：http://localhost:8080/getInfos3

☐ 自动重定向

☒ 跟随重定向

☒ 使用 KeepAlive

☐ 对POST使用multipart / form-data

参数 消息体数据 文件上传

名称：

第三步：重启服务并启动jmeter

```

2022-11-05 14:33:03.989 INFO 19332 --- [o-8080-exec-251] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.989 INFO 19332 --- [o-8080-exec-209] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.991 INFO 19332 --- [o-8080-exec-371] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.991 INFO 19332 --- [o-8080-exec-211] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.991 INFO 19332 --- [o-8080-exec-368] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.992 INFO 19332 --- [o-8080-exec-344] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.992 INFO 19332 --- [o-8080-exec-204] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.992 INFO 19332 --- [o-8080-exec-324] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.992 INFO 19332 --- [o-8080-exec-267] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.993 INFO 19332 --- [o-8080-exec-389] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.993 INFO 19332 --- [o-8080-exec-289] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:03.992 INFO 19332 --- [o-8080-exec-71] m.j.controller.TestController : 查询数据库3

```

```

2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-363] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-362] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-268] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-301] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-203] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-385] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-108] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.227 INFO 19332 --- [o-8080-exec-216] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.228 INFO 19332 --- [o-8080-exec-344] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.228 INFO 19332 --- [o-8080-exec-286] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.228 INFO 19332 --- [o-8080-exec-346] m.j.controller.TestController : 查询数据库3
2022-11-05 14:33:15.228 INFO 19332 --- [o-8080-exec-278] m.j.controller.TestController : 查询数据库3

```

38 / 117

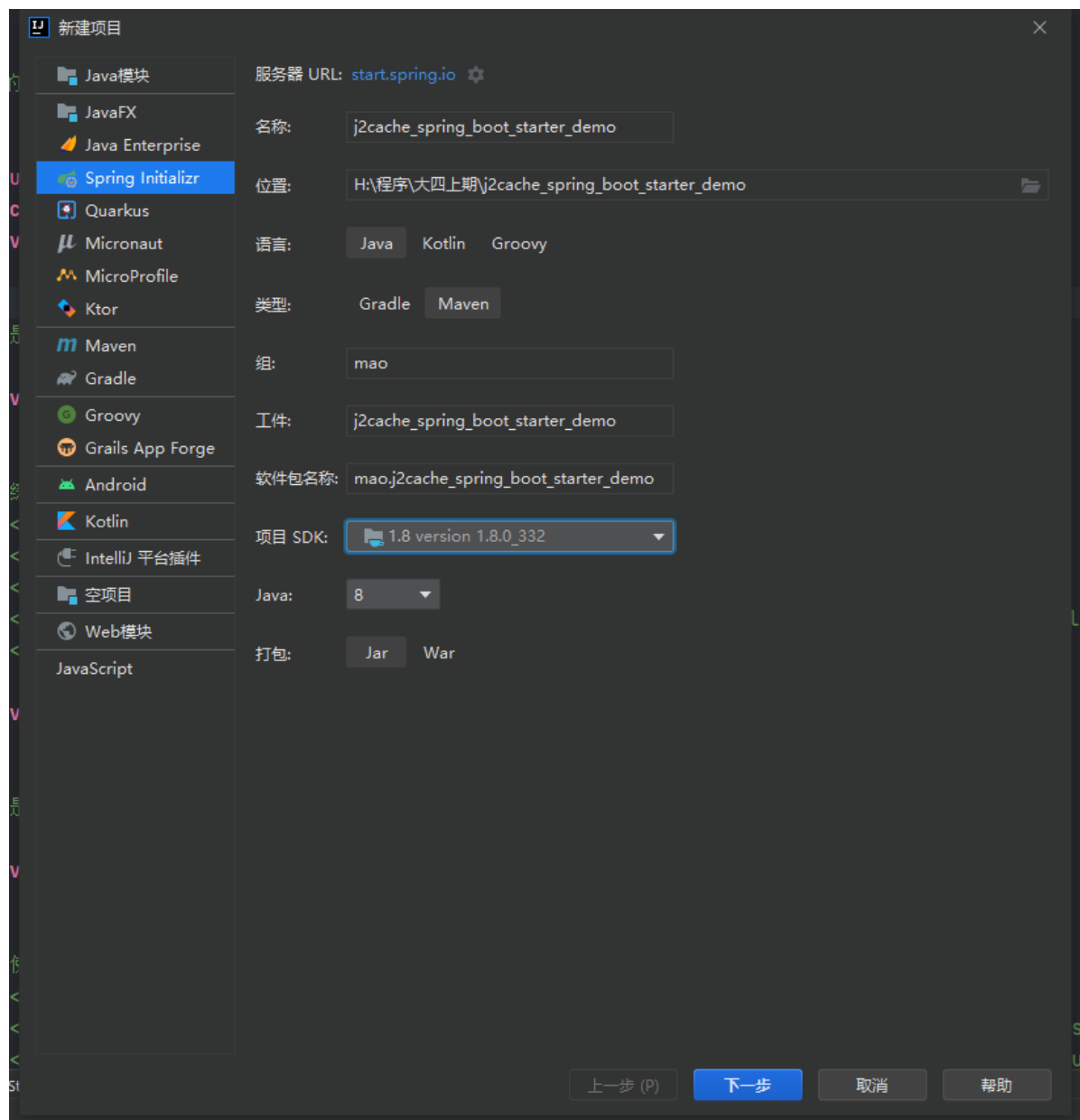
所以，使用缓存要小心，缓存击穿、缓存雪崩和缓存穿透需要自己解决

自定义spring boot starter

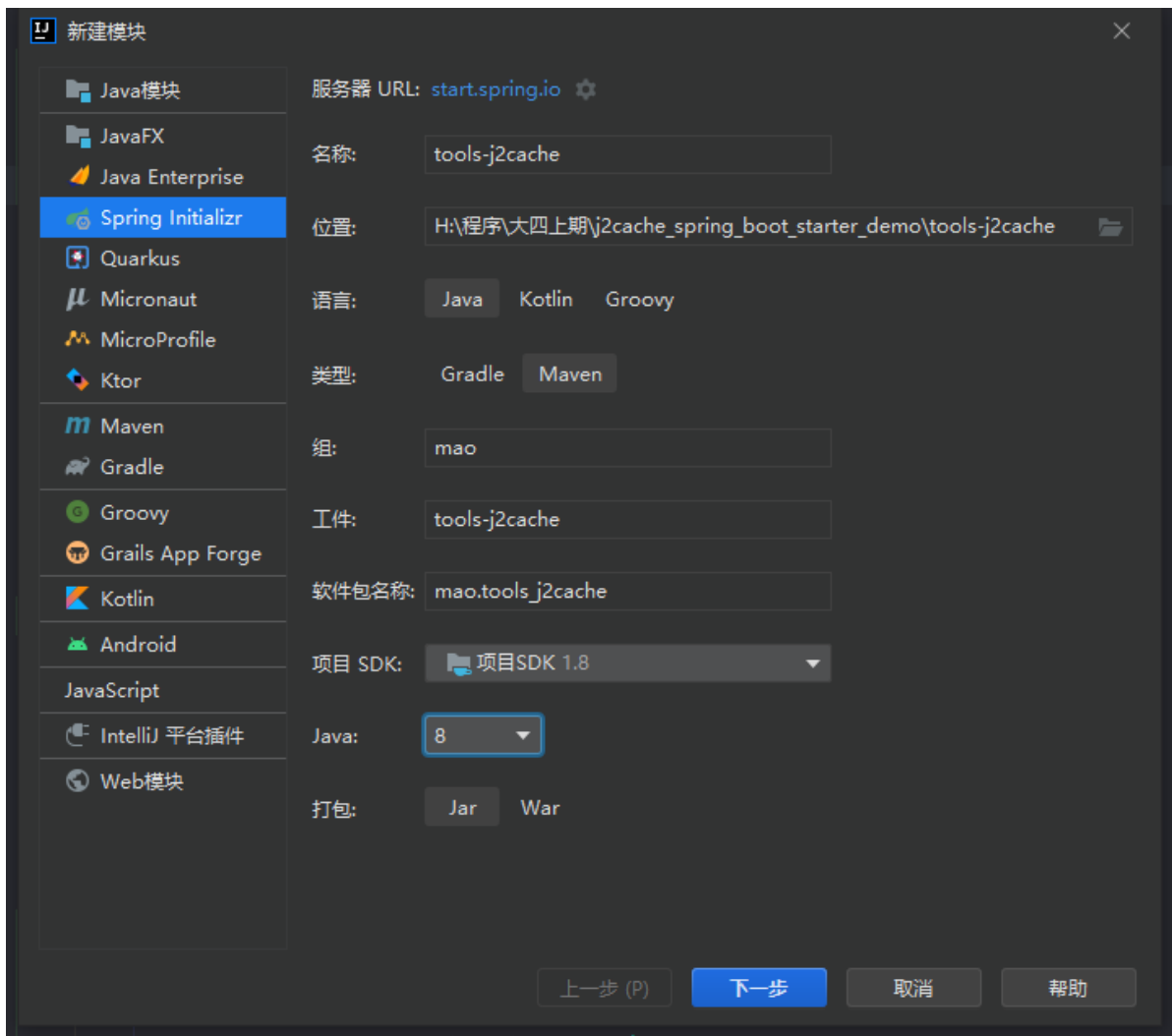
开发starter

第一步：初始化项目

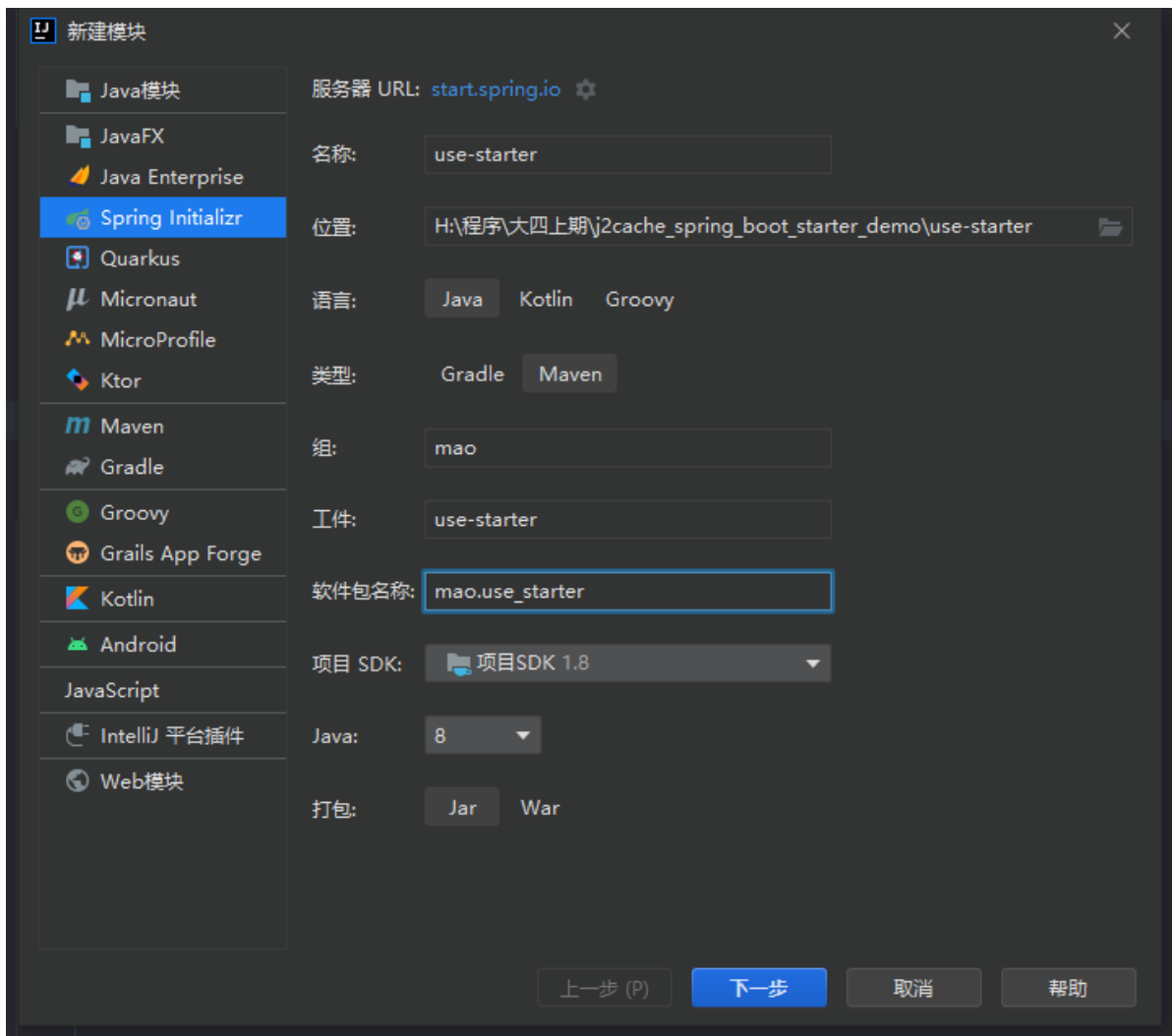
创建父工程j2cache_spring_boot_starter_demo



创建子工程tools-j2cache



创建子工程use-starter



第二步：修改pom文件

父工程j2cache_spring_boot_starter_demo的pom文件：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     https://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>2.7.1</version>
11        <relativePath/> <!-- lookup parent from repository -->
12    </parent>
13    <groupId>mao</groupId>
```

```

14     <artifactId>j2cache_spring_boot_starter_demo</artifactId>
15     <version>0.0.1-SNAPSHOT</version>
16     <name>j2cache_spring_boot_starter_demo</name>
17     <description>j2cache_spring_boot_starter_demo</description>
18     <packaging>pom</packaging>
19
20     <properties>
21         <java.version>1.8</java.version>
22     </properties>
23
24     <dependencies>
25
26 </dependencies>
27
28     <modules>
29         <module>tools-j2cache</module>
30         <module>use-starter</module>
31     </modules>
32
33     <dependencyManagement>
34         <dependencies>
35
36             <dependency>
37                 <groupId>net.oschina.j2cache</groupId>
38                 <artifactId>j2cache-spring-boot2-starter</artifactId>
39                 <version>2.8.0-release</version>
40             </dependency>
41
42             <dependency>
43                 <groupId>net.oschina.j2cache</groupId>
44                 <artifactId>j2cache-core</artifactId>
45                 <version>2.8.0-release</version>
46             </dependency>
47
48         </dependencies>
49     </dependencyManagement>
50
51     <build>
52         <plugins>
53             <plugin>
54                 <groupId>org.springframework.boot</groupId>
55                 <artifactId>spring-boot-maven-plugin</artifactId>
56             </plugin>
57         </plugins>
58     </build>
59
60 </project>
61

```

子工程tools-j2cache的pom文件:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

```

```

3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <artifactId>j2cache_spring_boot_starter_demo</artifactId>
7          <groupId>mao</groupId>
8          <version>0.0.1-SNAPSHOT</version>
9      </parent>
10     <artifactId>tools-j2cache</artifactId>
11     <name>tools-j2cache</name>
12     <description>tools-j2cache</description>
13
14     <properties>
15
16     </properties>
17
18     <dependencies>
19
20         <!--spring boot starter开发依赖-->
21         <dependency>
22             <groupId>org.springframework.boot</groupId>
23             <artifactId>spring-boot-starter</artifactId>
24         </dependency>
25
26         <dependency>
27             <groupId>org.springframework.boot</groupId>
28             <artifactId>spring-boot-autoconfigure</artifactId>
29         </dependency>
30
31         <dependency>
32             <groupId>org.springframework.boot</groupId>
33             <artifactId>spring-boot-configuration-processor</artifactId>
34         </dependency>
35
36         <dependency>
37             <groupId>org.springframework.boot</groupId>
38             <artifactId>spring-boot-starter-web</artifactId>
39         </dependency>
40
41         <dependency>
42             <groupId>net.oschina.j2cache</groupId>
43             <artifactId>j2cache-spring-boot2-starter</artifactId>
44         </dependency>
45
46         <dependency>
47             <groupId>net.oschina.j2cache</groupId>
48             <artifactId>j2cache-core</artifactId>
49             <exclusions>
50                 <exclusion>
51                     <groupId>org.slf4j</groupId>
52                     <artifactId>slf4j-simple</artifactId>
53                 </exclusion>
54                 <exclusion>
55                     <groupId>org.objenesis</groupId>
56                     <artifactId>objenesis</artifactId>
57                 </exclusion>
58                 <exclusion>
59                     <artifactId>javassist</artifactId>

```

```

60         <groupId>org.javassist</groupId>
61     </exclusion>
62     <exclusion>
63         <artifactId>fastjson</artifactId>
64         <groupId>com.alibaba</groupId>
65     </exclusion>
66 </exclusions>
67 </dependency>
68
69 <dependency>
70     <groupId>org.jgroups</groupId>
71     <artifactId>jgroups</artifactId>
72     <version>3.6.15.Final</version>
73 </dependency>
74
75 <dependency>
76     <artifactId>javassist</artifactId>
77     <groupId>org.javassist</groupId>
78     <version>3.25.0-GA</version>
79 </dependency>
80
81 <dependency>
82     <groupId>org.objenesis</groupId>
83     <artifactId>objenesis</artifactId>
84     <version>2.6</version>
85 </dependency>
86
87 <dependency>
88     <groupId>org.springframework</groupId>
89     <artifactId>spring-context</artifactId>
90     <scope>compile</scope>
91 </dependency>
92
93 <dependency>
94     <groupId>org.springframework</groupId>
95     <artifactId>spring-context-support</artifactId>
96     <scope>compile</scope>
97 </dependency>
98
99 <dependency>
100     <groupId>org.springframework.boot</groupId>
101     <artifactId>spring-boot-starter-data-redis</artifactId>
102     <scope>compile</scope>
103 </dependency>
104
105 <dependency>
106     <groupId>org.springframework</groupId>
107     <artifactId>spring-aspects</artifactId>
108 </dependency>
109 <dependency>
110     <groupId>org.aspectj</groupId>
111     <artifactId>aspectjrt</artifactId>
112     <version>1.9.2</version>
113 </dependency>
114 <dependency>
115     <groupId>org.aspectj</groupId>
116     <artifactId>aspectjweaver</artifactId>
117     <version>1.9.2</version>

```

```

118     </dependency>
119     <dependency>
120         <groupId>aopalliance</groupId>
121         <artifactId>aopalliance</artifactId>
122         <version>1.0</version>
123     </dependency>
124
125     <!--阿里巴巴的FastJson json解析-->
126     <dependency>
127         <groupId>com.alibaba</groupId>
128         <artifactId>fastjson</artifactId>
129         <version>1.2.79</version>
130     </dependency>
131
132     <dependency>
133         <groupId>cn.hutool</groupId>
134         <artifactId>hutool-all</artifactId>
135         <version>5.8.0</version>
136     </dependency>
137
138     <!--spring boot redisson 依赖-->
139     <dependency>
140         <groupId>org.redisson</groupId>
141         <artifactId>redisson-spring-boot-starter</artifactId>
142         <version>3.17.0</version>
143     </dependency>
144
145 </dependencies>
146
147 </project>
148

```

子工程use-starter的pom文件:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5  http://maven.apache.org/xsd/maven-4.0.0.xsd">
6      <modelVersion>4.0.0</modelVersion>
7      <parent>
8          <artifactId>j2cache_spring_boot_starter_demo</artifactId>
9          <groupId>mao</groupId>
10         <version>0.0.1-SNAPSHOT</version>
11     </parent>
12     <artifactId>use-starter</artifactId>
13     <name>use-starter</name>
14     <description>use-starter</description>
15
16     <properties>
17
18     </properties>

```

```

17
18     <dependencies>
19
20         <dependency>
21             <groupId>org.springframework.boot</groupId>
22             <artifactId>spring-boot-starter-web</artifactId>
23         </dependency>
24
25         <dependency>
26             <groupId>org.springframework.boot</groupId>
27             <artifactId>spring-boot-starter-test</artifactId>
28             <scope>test</scope>
29         </dependency>
30
31     </dependencies>
32
33     <build>
34         <plugins>
35             <plugin>
36                 <groupId>org.springframework.boot</groupId>
37                 <artifactId>spring-boot-maven-plugin</artifactId>
38             </plugin>
39         </plugins>
40     </build>
41
42 </project>

```

第三步：修改类J2CacheCache

```

1  package net.oschina.j2cache.cache.support;
2
3  import net.oschina.j2cache.CacheChannel;
4  import net.oschina.j2cache.CacheObject;
5  import net.oschina.j2cache.NullObject;
6  import org.springframework.cache.CacheManager;
7  import org.springframework.cache.support.AbstractValueAdaptingCache;
8  import org.springframework.cache.support.NullValue;
9
10 import java.util.concurrent.Callable;
11
12 /**
13  * {@link CacheManager} implementation for J2Cache.
14  */
15 public class J2CacheCache extends AbstractValueAdaptingCache
16 {
17
18     private CacheChannel cacheChannel;
19
20     private String j2CacheName = "j2cache";
21

```

```

22     public J2CacheCache(String cacheName, CacheChannel cacheChannel)
23     {
24         this(cacheName, cacheChannel, true);
25     }
26
27     public J2CacheCache(String cacheName, CacheChannel cacheChannel,
28 boolean allowNullValues)
29     {
30         super(allowNullValues);
31         j2CacheName = cacheName;
32         this.cacheChannel = cacheChannel;
33     }
34
35     @Override
36     public String getName()
37     {
38         return this.j2CacheName;
39     }
40
41     public void setJ2CacheName(String name)
42     {
43         this.j2CacheName = name;
44     }
45
46     @Override
47     public Object getNativeCache()
48     {
49         return this.cacheChannel;
50     }
51
52     @Override
53     public <T> T get(Object key, Callable<T> valueLoader)
54     {
55         T value;
56         try
57         {
58             value = valueLoader.call();
59         }
60         catch (Exception ex)
61         {
62             throw new ValueRetrievalException(key, valueLoader, ex);
63         }
64         put(key, value);
65         return value;
66     }
67
68     @Override
69     public void put(Object key, Object value)
70     {
71         cacheChannel.set(j2CacheName, String.valueOf(key), value,
72 super.isAllowNullValues());
73     }
74
75     @Override
76     public ValueWrapper putIfAbsent(Object key, Object value)
77     {
78         if (!cacheChannel.exists(j2CacheName, String.valueOf(key)))
79         {

```



```

78         cacheChannel.set(j2CacheName, String.valueOf(key), value);
79     }
80     return get(key);
81 }
82
83 @Override
84 public void evict(Object key)
85 {
86     cacheChannel.evict(j2CacheName, String.valueOf(key));
87 }
88
89 @Override
90 public void clear()
91 {
92     cacheChannel.clear(j2CacheName);
93 }
94
95 @Override
96 protected Object lookup(Object key)
97 {
98     CacheObject cacheObject = cacheChannel.get(j2CacheName,
99 String.valueOf(key));
100     if (cacheObject.rawValue() != null &&
101 cacheObject.rawValue().getClass().equals(NullObject.class) &&
102 super.isAllowNullValues())
103     {
104         return NullValue.INSTANCE;
105     }
106     return cacheObject.getValue();
107 }

```

第四步：修改类J2CacheCacheManger

```

1  package net.oschina.j2cache.cache.support;
2
3  import java.util.Collection;
4  import java.util.Collections;
5  import java.util.HashSet;
6  import java.util.LinkedHashSet;
7  import java.util.Set;
8
9  import net.oschina.j2cache.CacheChannel;
10 import org.springframework.cache.Cache;
11 import
org.springframework.cache.transaction.AbstractTransactionSupportingCacheManger;
12 import org.springframework.util.CollectionUtils;
13
14 import java.util.*;

```

```

15
16
17 /**
18  * {@link Cache} implementation for J2Cache.
19  */
20 public class J2CacheCacheManger extends
AbstractTransactionSupportingCacheManager
21 {
22
23     private boolean allowNullValues = true;
24
25     /**
26      * 缓存名称
27      */
28     private Collection<String> cacheNames;
29
30     private boolean dynamic = true;
31
32     private CacheChannel cacheChannel;
33
34     public J2CacheCacheManger(CacheChannel cacheChannel)
35     {
36         this.cacheChannel = cacheChannel;
37     }
38
39     /**
40      * 加载缓存
41      *
42      * @return {@link Collection}<{@link ?} {@link extends} {@link Cache}>
43      */
44     @Override
45     protected Collection<? extends Cache> loadCaches()
46     {
47         Collection<Cache> caches = new LinkedHashSet<>(cacheNames.size());
48         for (String name : cacheNames)
49         {
50             J2CacheCache cache = new J2CacheCache(name, cacheChannel,
allowNullValues);
51             caches.add(cache);
52         }
53         return caches;
54     }
55
56
57     /**
58      * 是允许空值
59      *
60      * @return boolean
61      */
62     public boolean isAllowNullValues()
63     {
64         return allowNullValues;
65     }
66
67     /**
68      * 设置允许空值
69      *
70      * @param allowNullValues 允许空值

```

```

71     */
72     public void setAllowNullValues(boolean allowNullValues)
73     {
74         this.allowNullValues = allowNullValues;
75     }
76
77     @Override
78     protected Cache getMissingCache(String name)
79     {
80         return this.dynamic ? new J2CacheCache(name, cacheChannel,
allowNullValues) : null;
81     }
82
83
84     /**
85      * 设置缓存名称
86      *
87      * @param cacheNames 缓存名称
88      */
89     public void setCacheNames(Collection<String> cacheNames)
90     {
91         Set<String> newCacheNames = CollectionUtils.isEmpty(cacheNames) ?
Collections.emptySet()
92             : new HashSet<>(cacheNames);
93         this.cacheNames = newCacheNames;
94         this.dynamic = newCacheNames.isEmpty();
95     }
96
97 }

```

第五步：添加类J2CacheSerializer

```

1  package net.oschina.j2cache.cache.support.util;
2
3  import net.oschina.j2cache.util.SerializationUtils;
4  import org.springframework.data.redis.serializer.RedisSerializer;
5  import org.springframework.data.redis.serializer.SerializationException;
6
7  import java.io.IOException;
8
9
10 public class J2CacheSerializer implements RedisSerializer<Object>
11 {
12
13     @Override
14     public byte[] serialize(Object t) throws SerializationException
15     {
16         try
17         {
18             return SerializationUtils.serialize(t);
19         }

```

```

20         catch (IOException e)
21         {
22             // TODO Auto-generated catch block
23             e.printStackTrace();
24         }
25         return null;
26     }
27
28     @Override
29     public Object deserialize(byte[] bytes) throws SerializationException
30     {
31         try
32         {
33             return SerializationUtils.deserialize(bytes);
34         }
35         catch (IOException e)
36         {
37             // TODO Auto-generated catch block
38             e.printStackTrace();
39         }
40         return null;
41     }
42
43 }

```

第六步：添加类SpringJ2CacheConfigUtil

```

1  package net.oschina.j2cache.cache.support.util;
2
3  import net.oschina.j2cache.J2CacheConfig;
4  import org.springframework.core.env.CompositePropertySource;
5  import org.springframework.core.env.EnumerablePropertySource;
6  import org.springframework.core.env.MapPropertySource;
7  import org.springframework.core.env.StandardEnvironment;
8
9  public class SpringJ2CacheConfigUtil
10 {
11
12     /**
13      * 从spring环境变量中查找j2cache配置
14      */
15     public static J2CacheConfig initFromConfig(StandardEnvironment
environment)
16     {
17         J2CacheConfig config = new J2CacheConfig();
18
19         config.setSerialization(environment.getProperty("j2cache.serialization"));
20         config.setBroadcast(environment.getProperty("j2cache.broadcast"));
21
22         config.setL1CacheName(environment.getProperty("j2cache.L1.provider_class"));
23     }
24 }

```

```

21     config.setL2CacheName(environment.getProperty("j2cache.L2.provider_class")
22 );
23
24     config.setSyncTtlToRedis(!"false".equalsIgnoreCase(environment.getProperty(
25 "j2cache.sync_ttl_to_redis")));
26
27     config.setDefaultCacheNullObject("true".equalsIgnoreCase(environment.getPr
28 operty("j2cache.default_cache_null_object")));
29     String l2_config_section =
30 environment.getProperty("j2cache.L2.config_section");
31     if (l2_config_section == null ||
32 l2_config_section.trim().equals(""))
33     {
34         l2_config_section = config.getL2CacheName();
35     }
36     String l2_section = l2_config_section;
37     //配置在 application.yml 或者 j2cache.properties 中时，这里能正常读取
38     environment.getPropertySources().forEach(a ->
39     {
40         if (a instanceof MapPropertySource)
41         {
42             MapPropertySource c = (MapPropertySource) a;
43             c.getSource().forEach((k, v) ->
44             {
45                 String key = k;
46                 if (key.startsWith(config.getBroadcast() + "."))
47                 {
48                     config.getBroadcastProperties().setProperty(key.substring((config.getBroad
49 cast() + ".").length()),
50                         environment.getProperty(key));
51                 }
52                 if (key.startsWith(config.getL1CacheName() + "."))
53                 {
54                     config.getL1CacheProperties().setProperty(key.substring((config.getL1Cache
55 Name() + ".").length()),
56                         environment.getProperty(key));
57                 }
58                 if (key.startsWith(l2_section + "."))
59                 {
60                     config.getL2CacheProperties().setProperty(key.substring((l2_section +
61 ".").length()),
62                         environment.getProperty(key));
63                 }
64             });
65         }
66     });
67
68     //配置在 nacos 中时，上面那段代码无法获取配置
69     if (config.getL1CacheProperties().isEmpty() ||
70 config.getL2CacheProperties().isEmpty() ||
71 config.getBroadcastProperties().isEmpty())
72     {
73         environment.getPropertySources().forEach(ps ->
74         {

```

```

63         String[] propertyNames = new String[]{};
64         if (ps instanceof CompositePropertySource)
65         {
66             CompositePropertySource cps = (CompositePropertySource)
ps;
67             propertyNames = cps.getPropertyNames();
68         }
69         else if (ps instanceof EnumerablePropertySource)
70         {
71             EnumerablePropertySource eps =
(EnumerablePropertySource) ps;
72             propertyNames = eps.getPropertyNames();
73         }
74         setProperty(config, environment, l2_section,
propertyNames);
75     });
76 }
77     return config;
78 }
79
80     private static void setProperty(J2CacheConfig config,
StandardEnvironment environment, String l2_section, String[] propertyNames)
81     {
82         for (String key : propertyNames)
83         {
84             if (key.startsWith(config.getBroadcast() + "."))
85             {
86
87                 config.getBroadcastProperties().setProperty(key.substring((config.getBroad
cast() + ".").length()),
88                     environment.getProperty(key));
89             }
90             if (key.startsWith(config.getL1CacheName() + "."))
91             {
92                 config.getL1CacheProperties().setProperty(key.substring((config.getL1Cache
Name() + ".").length()),
93                     environment.getProperty(key));
94             }
95             if (key.startsWith(l2_section + "."))
96             {
97                 config.getL2CacheProperties().setProperty(key.substring((l2_section +
98                 ".").length()),
99                     environment.getProperty(key));
100             }
101         }
102     }

```

第七步：添加类SpringUtil

```
1 package net.oschina.j2cache.cache.support.util;
2
3 import org.springframework.beans.BeansException;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.ApplicationContextAware;
6
7 /**
8  * spring 工具类
9  */
10 public class SpringUtil implements ApplicationContextAware
11 {
12
13     /**
14      * 应用程序上下文
15      */
16     private static ApplicationContext applicationContext;
17
18     /**
19      * 获取applicationContext
20      */
21     public static ApplicationContext getApplicationContext()
22     {
23         return applicationContext;
24     }
25
26     @Override
27     public void setApplicationContext(ApplicationContext applicationContext)
28     throws BeansException
29     {
30         if
31 (net.oschina.j2cache.cache.support.util.SpringUtil.applicationContext ==
32 null)
33     {
34         net.oschina.j2cache.cache.support.util.SpringUtil.applicationContext =
35 applicationContext;
36     }
37
38     /**
39      * 通过name获取 Bean.
40      */
41     public static Object getBean(String name)
42     {
43         return getApplicationContext().getBean(name);
44     }
45
46     /**
47      * 通过class获取Bean.
48      */
49     public static <T> T getBean(Class<T> clazz)
50     {
51         return getApplicationContext().getBean(clazz);
52     }
53 }
```

```

49     }
50
51     /**
52      * 通过name,以及Clazz返回指定的Bean
53      */
54     public static <T> T getBean(String name, Class<T> clazz)
55     {
56         return getApplicationContext().getBean(name, clazz);
57     }
58
59 }

```

第八步：添加类ConfigureNotifyKeyspaceEventsAction

```

1  package net.oschina.j2cache.cache.support.redis;
2
3  import org.springframework.dao.InvalidDataAccessApiUsageException;
4  import org.springframework.data.redis.connection.RedisConnection;
5
6  import java.util.Properties;
7
8  /**
9   * 设置redis键值回调
10  */
11  public class ConfigureNotifyKeyspaceEventsAction
12  {
13
14      /**
15       * 配置用于事件通知
16       */
17      static final String CONFIG_NOTIFY_KEYSPACE_EVENTS = "notify-keyspace-
events";
18
19
20      /**
21       * 配置
22       *
23       * @param connection 连接
24       */
25      public void config(RedisConnection connection)
26      {
27          String notifyOptions = getNotifyOptions(connection);
28          String customizedNotifyOptions = notifyOptions;
29          if (!customizedNotifyOptions.contains("E"))
30          {
31              customizedNotifyOptions += "E";
32          }
33          boolean A = customizedNotifyOptions.contains("A");
34          if (!(A || customizedNotifyOptions.contains("g")))
35          {
36              customizedNotifyOptions += "g";

```



```

37     }
38     if (!(A || customizedNotifyOptions.contains("x")))
39     {
40         customizedNotifyOptions += "x";
41     }
42     if (!notifyOptions.equals(customizedNotifyOptions))
43     {
44         connection.setConfig(CONFIG_NOTIFY_KEYSPACE_EVENTS,
customizedNotifyOptions);
45     }
46 }
47
48 /**
49  * 得到通知选项
50  *
51  * @param connection 连接
52  * @return {@link String}
53  */
54 private String getNotifyOptions(RedisConnection connection)
55 {
56     try
57     {
58         Properties config =
connection.getConfig(CONFIG_NOTIFY_KEYSPACE_EVENTS);
59         if (config.isEmpty())
60         {
61             return "";
62         }
63         return
config.getProperty(config.stringPropertyNames().iterator().next());
64     }
65     catch (InvalidDataAccessApiUsageException e)
66     {
67         throw new IllegalStateException(
68             "Unable to configure Redis to keyspace notifications.
See http://docs.spring.io/spring-session/docs/current/reference/html5/#api-redisoperationssessionrepository-sessiondestroyedevent",
69             e);
70     }
71 }
72 }

```

第九步：添加类SpringRedisActiveMessageListener

```

1 package net.oschina.j2cache.cache.support.redis;
2
3 import net.oschina.j2cache.cluster.ClusterPolicy;
4 import org.slf4j.Logger;
5 import org.slf4j.LoggerFactory;
6 import org.springframework.data.redis.connection.Message;
7 import org.springframework.data.redis.connection.MessageListener;

```

```

8
9  /**
10  * 监听二缓key失效，主动清除本地缓存
11  */
12  public class SpringRedisActiveMessageListener implements MessageListener
13  {
14
15      /**
16      * 日志记录器
17      */
18      private static Logger logger =
19      LoggerFactory.getLogger(net.oschina.j2cache.cache.support.redis.SpringRedisA
20      ctiveMessageListener.class);
21
22      /**
23      * 集群政策
24      */
25      private ClusterPolicy clusterPolicy;
26
27      /**
28      * 名称空间
29      */
30      private String namespace;
31
32      SpringRedisActiveMessageListener(ClusterPolicy clusterPolicy, String
33      namespace)
34      {
35
36          this.clusterPolicy = clusterPolicy;
37          this.namespace = namespace;
38      }
39
40      @Override
41      public void onMessage(Message message, byte[] pattern)
42      {
43          String key = message.toString();
44          if (key == null)
45          {
46              return;
47          }
48          if (key.startsWith(namespace + ":"))
49          {
50              String[] k = key.replaceFirst(namespace + ":", "").split(":",
51              2);
52              if (k.length != 2)
53              {
54                  return;
55              }
56              clusterPolicy.evict(k[0], k[1]);
57          }
58      }
59  }

```

第十步：添加类SpringRedisCache

```
1 package net.oschina.j2cache.cache.support.redis;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5 import java.util.Collection;
6 import java.util.List;
7 import java.util.Map;
8 import java.util.Set;
9
10 import net.oschina.j2cache.Level2Cache;
11 import org.springframework.data.redis.core.RedisCallback;
12 import org.springframework.data.redis.core.RedisTemplate;
13
14 /**
15  * 重新实现二级缓存，采用hash结构缓存数据
16  */
17 public class SpringRedisCache implements Level2Cache
18 {
19
20     /**
21      * 名称空间
22      */
23     private String namespace;
24
25     /**
26      * 地区
27      */
28     private String region;
29
30     private RedisTemplate<String, Serializable> redisTemplate;
31
32     public SpringRedisCache(String namespace, String region,
33 RedisTemplate<String, Serializable> redisTemplate)
34     {
35         if (region == null || region.isEmpty())
36         {
37             region = "_"; // 缺省region
38         }
39         this.namespace = namespace;
40         this.redisTemplate = redisTemplate;
41         this.region = getRegionName(region);
42     }
43
44     private String getRegionName(String region)
45     {
46         if (namespace != null && !namespace.isEmpty())
47         {
48             region = namespace + ":" + region;
49         }
50         return region;
51     }
52
53     @Override
```

```

53     public void clear()
54     {
55         redisTemplate.opsForHash().delete(region);
56     }
57
58     @Override
59     public boolean exists(String key)
60     {
61         return redisTemplate.opsForHash().hasKey(region, key);
62     }
63
64     @Override
65     public void evict(String... keys)
66     {
67         for (String k : keys)
68         {
69             if (!k.equals("null"))
70             {
71                 redisTemplate.opsForHash().delete(region, k);
72             }
73             else
74             {
75                 redisTemplate.delete(region);
76             }
77         }
78     }
79
80     @Override
81     public Collection<String> keys()
82     {
83         Set<Object> list = redisTemplate.opsForHash().keys(region);
84         List<String> keys = new ArrayList<>(list.size());
85         for (Object object : list)
86         {
87             keys.add((String) object);
88         }
89         return keys;
90     }
91
92     @Override
93     public byte[] getBytes(String key)
94     {
95         return
redisTemplate.opsForHash().getOperations().execute((RedisCallback<byte[]>)
redis ->
96             redis.hGet(region.getBytes(), key.getBytes()));
97     }
98
99     @Override
100    public List<byte[]> getBytes(Collection<String> keys)
101    {
102        return
redisTemplate.opsForHash().getOperations().execute((RedisCallback<List<byte
[]>>) redis ->
103        {
104            byte[][] bytes = keys.stream().map(k ->
k.getBytes()).toArray(byte[][]::new);
105            return redis.hmGet(region.getBytes(), bytes);

```

```

106     });
107 }
108
109 @Override
110 public void put(String key, Object value)
111 {
112     redisTemplate.opsForHash().put(region, key, value);
113 }
114
115 /**
116  * 设置缓存数据的有效期
117  */
118 @Override
119 public void put(String key, Object value, long timeToLiveInSeconds)
120 {
121     redisTemplate.opsForHash().put(region, key, value);
122 }
123
124 @Override
125 public void setBytes(String key, byte[] bytes)
126 {
127
128     redisTemplate.opsForHash().getOperations().execute((RedisCallback<List<byte
129     e[]>>) redis ->
130     {
131         redis.set(_key(key).getBytes(), bytes);
132         redis.hSet(region.getBytes(), key.getBytes(), bytes);
133         return null;
134     });
135 }
136
137 @Override
138 public void setBytes(Map<String, byte[]> bytes)
139 {
140     bytes.forEach((k, v) ->
141     {
142         setBytes(k, v);
143     });
144 }
145
146 private String _key(String key)
147 {
148     return this.region + ":" + key;
149 }

```

第十一步：添加类SpringRedisGenericCache

```

1 package net.oschina.j2cache.cache.support.redis;
2

```

```

3  import java.io.Serializable;
4  import java.io.UnsupportedEncodingException;
5  import java.util.Collection;
6  import java.util.List;
7  import java.util.Map;
8  import java.util.stream.Collectors;
9
10 import net.oschina.j2cache.Level2Cache;
11 import org.slf4j.Logger;
12 import org.slf4j.LoggerFactory;
13 import org.springframework.data.redis.core.RedisCallback;
14 import org.springframework.data.redis.core.RedisTemplate;
15
16 public class SpringRedisGenericCache implements Level2Cache
17 {
18
19     /**
20      * 日志
21      */
22     private final static Logger log =
23         LoggerFactory.getLogger(net.oschina.j2cache.cache.support.redis.SpringRedis
24             GenericCache.class);
25
26     /**
27      * 名称空间
28      */
29     private String namespace;
30
31     private String region;
32
33     private RedisTemplate<String, Serializable> redisTemplate;
34
35     public SpringRedisGenericCache(String namespace, String region,
36         RedisTemplate<String, Serializable> redisTemplate)
37     {
38         if (region == null || region.isEmpty())
39         {
40             region = "_"; // 缺省region
41         }
42         this.namespace = namespace;
43         this.redisTemplate = redisTemplate;
44         this.region = getRegionName(region);
45     }
46
47     private String getRegionName(String region)
48     {
49         if (namespace != null && !namespace.isEmpty())
50         {
51             region = namespace + ":" + region;
52         }
53         return region;
54     }
55
56     @Override
57     public void clear()
58     {
59         collection<String> keys = keys();
60         keys.stream().forEach(k ->

```

```

58         {
59             redisTemplate.delete(this.region + ":" + k);
60         });
61     }
62
63     @Override
64     public boolean exists(String key)
65     {
66         return redisTemplate.execute((RedisCallback<Boolean>) redis ->
67         {
68             return redis.exists(_key(key));
69         });
70     }
71
72     @Override
73     public void evict(String... keys)
74     {
75         for (String k : keys)
76         {
77             redisTemplate.execute((RedisCallback<Long>) redis ->
78             {
79                 return redis.del(_key(k));
80             });
81         }
82     }
83
84     @Override
85     public Collection<String> keys()
86     {
87         return redisTemplate.keys(this.region + ":*").stream().map(k ->
88         k.substring(this.region.length() +
89 1)).collect(Collectors.toSet());
90     }
91
92     @Override
93     public byte[] getBytes(String key)
94     {
95         return redisTemplate.execute((RedisCallback<byte[]>) redis ->
96         {
97             return redis.get(_key(key));
98         });
99     }
100
101     @Override
102     public List<byte[]> getBytes(Collection<String> keys)
103     {
104         return redisTemplate.execute((RedisCallback<List<byte[]>>) redis ->
105         {
106             byte[][] bytes = keys.stream().map(k -> _key(k)).toArray(byte[]
107             []::new);
108             return redis.mGet(bytes);
109         });
110     }
111
112     @Override
113     public void setBytes(String key, byte[] bytes, long
timeToLiveInSeconds)
114     {

```

```

113         if (timeToLiveInSeconds <= 0)
114         {
115             log.debug(String.format("Invalid timeToLiveInSeconds value : %d
, skipped it.", timeToLiveInSeconds));
116             setBytes(key, bytes);
117         }
118         else
119         {
120             redisTemplate.execute((RedisCallback<List<byte[]>>) redis ->
121             {
122                 redis.setEx(_key(key), (int) timeToLiveInSeconds, bytes);
123                 return null;
124             });
125         }
126     }
127
128     @Override
129     public void setBytes(Map<String, byte[]> bytes, long
timeToLiveInSeconds)
130     {
131         bytes.forEach((k, v) -> setBytes(k, v, timeToLiveInSeconds));
132     }
133
134     @Override
135     public void setBytes(String key, byte[] bytes)
136     {
137         redisTemplate.execute((RedisCallback<byte[]>) redis ->
138         {
139             redis.set(_key(key), bytes);
140             return null;
141         });
142     }
143
144     @Override
145     public void setBytes(Map<String, byte[]> bytes)
146     {
147         bytes.forEach((k, v) -> setBytes(k, v));
148     }
149
150     private byte[] _key(String key)
151     {
152         byte[] k;
153         try
154         {
155             k = (this.region + ":" + key).getBytes("utf-8");
156         }
157         catch (UnsupportedEncodingException e)
158         {
159             e.printStackTrace();
160             k = (this.region + ":" + key).getBytes();
161         }
162         return k;
163     }
164 }

```


第十二步：添加类SpringRedisMessageListener

```
1 package net.oschina.j2cache.cache.support.redis;
2
3 import net.oschina.j2cache.Command;
4 import net.oschina.j2cache.cluster.ClusterPolicy;
5 import net.oschina.j2cache.util.SerializationUtils;
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import org.springframework.data.redis.connection.Message;
9 import org.springframework.data.redis.connection.MessageListener;
10
11 /**
12  * spring redis 订阅消息监听
13  */
14 public class SpringRedisMessageListener implements MessageListener
15 {
16
17     /**
18      * 日志记录器
19      */
20     private static Logger logger =
21         LoggerFactory.getLogger(net.oschina.j2cache.cache.support.redis.SpringRedisMessageListener.class);
22
23     /**
24      * 本地命令id
25      */
26     private int LOCAL_COMMAND_ID = Command.genRandomSrc(); //命令源标识，随机生成，每个节点都有唯一标识
27
28     /**
29      * 集群政策
30      */
31     private ClusterPolicy clusterPolicy;
32
33     /**
34      * 通道
35      */
36     private String channel;
37
38     SpringRedisMessageListener(ClusterPolicy clusterPolicy, String channel)
39     {
40         this.clusterPolicy = clusterPolicy;
41         this.channel = channel;
42     }
43
44     private boolean isLocalCommand(Command cmd)
45     {
46         return cmd.getSrc() == LOCAL_COMMAND_ID;
47     }
48
49     @Override
50     public void onMessage(Message message, byte[] pattern)
51     {
52         byte[] messageChannel = message.getChannel();
```

```

51     byte[] messageBody = message.getBody();
52     if (messageChannel == null || messageBody == null)
53     {
54         return;
55     }
56     try
57     {
58         Command cmd =
Command.parse(String.valueOf(SerializationUtils.deserialize(messageBody)));
59         if (cmd == null || isLocalCommand(cmd))
60         {
61             return;
62         }
63
64         switch (cmd.getOperator())
65         {
66             case Command.OPT_JOIN:
67                 logger.info("Node-" + cmd.getSrc() + " joined to " +
this.channel);
68                 break;
69             case Command.OPT_EVICT_KEY:
70                 clusterPolicy.evict(cmd.getRegion(), cmd.getKeys());
71                 logger.debug("Received cache evict message, region=" +
cmd.getRegion() + ",key=" + String.join(",", cmd.getKeys()));
72                 break;
73             case Command.OPT_CLEAR_KEY:
74                 clusterPolicy.clear(cmd.getRegion());
75                 logger.debug("Received cache clear message, region=" +
cmd.getRegion());
76                 break;
77             case Command.OPT_QUIT:
78                 logger.info("Node-" + cmd.getSrc() + " quit to " +
this.channel);
79                 break;
80             default:
81                 logger.warn("Unknown message type = " +
cmd.getOperator());
82         }
83     }
84     catch (Exception e)
85     {
86         logger.error("Failed to handle received msg", e);
87     }
88 }
89
90 }

```

第十三步：添加类SpringRedisProvider

```

1 package net.oschina.j2cache.cache.support.redis;
2

```

```

3 import java.io.Serializable;
4 import java.util.Collection;
5 import java.util.Collections;
6 import java.util.Properties;
7 import java.util.concurrent.ConcurrentHashMap;
8
9 import net.oschina.j2cache.Cache;
10 import net.oschina.j2cache.CacheChannel;
11 import net.oschina.j2cache.CacheExpiredListener;
12 import net.oschina.j2cache.CacheObject;
13 import net.oschina.j2cache.CacheProvider;
14 import net.oschina.j2cache.NullCache;
15 import net.oschina.j2cache.cache.support.util.SpringUtil;
16 import org.springframework.data.redis.core.RedisTemplate;
17
18 /**
19  * spring redis缓存
20  */
21 public class SpringRedisProvider implements CacheProvider
22 {
23
24     /**
25      * 缓存
26      */
27     protected ConcurrentHashMap<String, Cache> caches = new
ConcurrentHashMap<>();
28     private RedisTemplate<String, Serializable> redisTemplate;
29     /**
30      * 配置
31      */
32     private net.oschina.j2cache.autoconfigure.J2CacheConfig config;
33     /**
34      * 名称空间
35      */
36     private String namespace;
37     /**
38      * 存储
39      */
40     private String storage;
41
42     @Override
43     public String name()
44     {
45         return "redis";
46     }
47
48     @Override
49     public int level()
50     {
51         return CacheObject.LEVEL_2;
52     }
53
54     @Override
55     public Collection<CacheChannel.Region> regions()
56     {
57         return Collections.emptyList();
58     }
59

```

```

60  /**
61   * 建立缓存
62   *
63   * @param region 地区
64   * @param listener 侦听器
65   * @return {@link Cache}
66   */
67  @Override
68  public Cache buildCache(String region, CacheExpiredListener listener)
69  {
70      if (config.getL2CacheOpen() == false)
71      {
72          return new NullCache();
73      }
74      Cache cache = caches.get(region);
75      if (cache == null)
76      {
77          synchronized
78      (net.oschina.j2cache.cache.support.redis.SpringRedisProvider.class)
79      {
80          cache = caches.get(region);
81          if (cache == null)
82          {
83              if ("hash".equalsIgnoreCase(this.storage))
84              {
85                  cache = new SpringRedisCache(this.namespace,
86                  region, redisTemplate);
87              }
88              else
89              {
90                  cache = new SpringRedisGenericCache(this.namespace,
91                  region, redisTemplate);
92              }
93              caches.put(region, cache);
94          }
95          return cache;
96      }
97      @Override
98      public Cache buildCache(String region, long timeToLiveInSeconds,
99      CacheExpiredListener listener)
100      {
101          return buildCache(region, listener);
102      }
103      @SuppressWarnings("unchecked")
104      @Override
105      public void start(Properties props)
106      {
107          this.namespace = props.getProperty("namespace");
108          this.storage = props.getProperty("storage");
109          this.config =
110      SpringUtil.getBean(net.oschina.j2cache.autoconfigure.J2CacheConfig.class);
111          if (config.getL2CacheOpen() == false)
112          {
113              return;

```

```

113     }
114     this.redisTemplate = SpringUtil.getBean("j2CacheRedisTemplate",
RedisTemplate.class);
115 }
116
117 @Override
118 public void stop()
119 {
120     // 由spring控制
121 }
122
123 }

```

第十四步：添加类SpringRedisPubSubPolicy

```

1  package net.oschina.j2cache.cache.support.redis;
2
3  import java.io.Serializable;
4  import java.util.ArrayList;
5  import java.util.List;
6  import java.util.Properties;
7
8  import net.oschina.j2cache.CacheProviderHolder;
9  import net.oschina.j2cache.Command;
10 import net.oschina.j2cache.J2CacheConfig;
11 import net.oschina.j2cache.cache.support.util.SpringUtil;
12 import net.oschina.j2cache.cluster.ClusterPolicy;
13 import org.springframework.data.redis.core.RedisTemplate;
14 import org.springframework.data.redis.listener.PatternTopic;
15 import
org.springframework.data.redis.listener.RedisMessageListenerContainer;
16
17 /**
18  * 使用spring redis实现订阅功能
19  */
20 public class SpringRedisPubSubPolicy implements ClusterPolicy
21 {
22
23     /**
24      * 是否是主动模式
25      */
26     private static boolean isActive = false;
27     private int LOCAL_COMMAND_ID = Command.genRandomSrc(); //命令源标识，随机
生成，每个节点都有唯一标识
28     private RedisTemplate<String, Serializable> redisTemplate;
29     private net.oschina.j2cache.autoconfigure.J2CacheConfig config;
30     private CacheProviderHolder holder;
31     private String channel = "j2cache_channel";
32
33     @Override
34     public boolean isLocalCommand(Command cmd)

```

```

35     {
36         return cmd.getSrc() == LOCAL_COMMAND_ID;
37     }
38
39     @SuppressWarnings("unchecked")
40     @Override
41     public void connect(Properties props, CacheProviderHolder holder)
42     {
43         this.holder = holder;
44         this.config =
SpringUtil.getBean(net.oschina.j2cache.autoconfigure.J2CacheConfig.class);
45         if (config.getL2CacheOpen() == false)
46         {
47             return;
48         }
49         J2CacheConfig j2config = SpringUtil.getBean(J2CacheConfig.class);
50         this.redisTemplate = SpringUtil.getBean("j2CacheRedisTemplate",
RedisTemplate.class);
51         String channel_name =
j2config.getL2CacheProperties().getProperty("channel");
52         if (channel_name != null && !channel_name.isEmpty())
53         {
54             this.channel = channel_name;
55         }
56         RedisMessageListenerContainer listenerContainer =
SpringUtil.getBean("j2CacheRedisMessageListenerContainer",
RedisMessageListenerContainer.class);
57         String namespace =
j2config.getL2CacheProperties().getProperty("namespace");
58         String database =
j2config.getL2CacheProperties().getProperty("database");
59         String expired = "__keyevent@" + (database == null ||
"".equals(database) ? "0" : database) + ".__:expired";
60         String del = "__keyevent@" + (database == null ||
"".equals(database) ? "0" : database) + ".__:del";
61         List<PatternTopic> topics = new ArrayList<>();
62         topics.add(new PatternTopic(expired));
63         topics.add(new PatternTopic(del));
64
65         if ("active".equals(config.getCacheCleanMode()))
66         {
67             isActive = true;
68             //设置键值回调 需要redis支持键值回调
69             ConfigureNotifyKeyspaceEventsAction action = new
ConfigureNotifyKeyspaceEventsAction();
70
71             action.config(listenerContainer.getConnectionFactory().getConnection());
72             listenerContainer.addListener(new
SpringRedisActiveMessageListener(this, namespace), topics);
73         }
74         else if ("blend".equals(config.getCacheCleanMode()))
75         {
76             //设置键值回调 需要redis支持键值回调
77             ConfigureNotifyKeyspaceEventsAction action = new
ConfigureNotifyKeyspaceEventsAction();
78
79             action.config(listenerContainer.getConnectionFactory().getConnection());

```

```

78         listenerContainer.addMessageListener(new
SpringRedisActiveMessageListener(this, namespace), topics);
79         listenerContainer.addMessageListener(new
SpringRedisMessageListener(this, this.channel), new
PatternTopic(this.channel));
80     }
81     else
82     {
83         listenerContainer.addMessageListener(new
SpringRedisMessageListener(this, this.channel), new
PatternTopic(this.channel));
84     }
85
86 }
87
88 /**
89  * 删除本地某个缓存条目
90  *
91  * @param region 区域名称
92  * @param keys   缓存键值
93  */
94 public void evict(String region, String... keys)
95 {
96     holder.getLevel1Cache(region).evict(keys);
97 }
98
99 /**
100  * 清除本地整个缓存区域
101  *
102  * @param region 区域名称
103  */
104 public void clear(String region)
105 {
106     holder.getLevel1Cache(region).clear();
107 }
108
109 /**
110  * 发布
111  *
112  * @param cmd cmd
113  */
114 @Override
115 public void publish(Command cmd)
116 {
117     if (!isActive && config.getL2CacheOpen())
118     {
119         cmd.setSrc(LOCAL_COMMAND_ID);
120         redisTemplate.convertAndSend(this.channel, cmd.json());
121     }
122 }
123
124 /**
125  * 断开连接
126  */
127 @Override
128 public void disconnect()
129 {
130     if (!isActive && config.getL2CacheOpen())

```

```

131         {
132             Command cmd = new Command();
133             cmd.setSrc(LOCAL_COMMAND_ID);
134             cmd.setOperator(Command.OPT_QUIT);
135             redisTemplate.convertAndSend(this.channel, cmd.json());
136         }
137     }
138
139 }

```

第十五步：添加配置类J2CacheAutoConfiguration

```

1  package net.oschina.j2cache.autoconfigure;
2
3  import net.oschina.j2cache.CacheChannel;
4  import net.oschina.j2cache.J2Cache;
5  import net.oschina.j2cache.J2CacheBuilder;
6  import net.oschina.j2cache.cache.support.util.SpringJ2CacheConfigUtil;
7  import net.oschina.j2cache.cache.support.util.SpringUtil;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
10 import
11 org.springframework.boot.context.properties.EnableConfigurationProperties;
12 import org.springframework.context.annotation.Bean;
13 import org.springframework.context.annotation.Configuration;
14 import org.springframework.context.annotation.PropertySource;
15 import org.springframework.core.env.StandardEnvironment;
16
17 import java.io.IOException;
18
19 /**
20  * 启动入口
21  */
22 @ConditionalOnClass(J2Cache.class)
23 @EnableConfigurationProperties({J2CacheConfig.class})
24 @Configuration
25 @PropertySource(value = "${j2cache.config-location}", encoding = "UTF-8",
26 ignoreResourceNotFound = true)
27 public class J2CacheAutoConfiguration
28 {
29     @Autowired
30     private StandardEnvironment standardEnvironment;
31
32     @Bean
33     public net.oschina.j2cache.J2CacheConfig j2CacheConfig() throws
34 IOException
35     {
36         net.oschina.j2cache.J2CacheConfig cacheConfig =
37 SpringJ2CacheConfigUtil.initFromConfig(standardEnvironment);

```



```

36         return cacheConfig;
37     }
38
39     @Bean
40     @DependsOn({"springUtil", "j2CacheConfig"})
41     public CacheChannel cacheChannel(net.oschina.j2cache.J2CacheConfig
j2CacheConfig) throws IOException
42     {
43         J2CacheBuilder builder = J2CacheBuilder.init(j2CacheConfig);
44         return builder.getChannel();
45     }
46
47     @Bean
48     public SpringUtil springUtil()
49     {
50         return new SpringUtil();
51     }
52
53 }

```

第十六步：添加配置属性类J2CacheConfig

```

1  package net.oschina.j2cache.autoconfigure;
2
3  import org.springframework.boot.context.properties.ConfigurationProperties;
4
5  /**
6   * 相关的配置信息
7   */
8  @ConfigurationProperties(prefix = "j2cache")
9  public class J2CacheConfig
10 {
11     private String configLocation = "/j2cache.properties";
12
13     /**
14      * 是否开启spring cache缓存,注意:开启后需要添加spring.cache.type=GENERIC,将
缓存类型设置为GENERIC
15      */
16     private Boolean openSpringCache = false;
17
18     /**
19      * 缓存清除模式,
20      * <ul>
21      * <li>active:主动清除, 二级缓存过期主动通知各节点清除, 优点在于所有节点可以同时收
到缓存清除</li>
22      * <li>passive:被动清除, 一级缓存过期进行通知各节点清除一二级缓存, </li>
23      * <li>blend:两种模式一起运作, 对于各个节点缓存准确以及及时性要求高的可以使用, 正
常用前两种模式中一个就可</li>
24      * </ul>
25      */
26     private String cacheCleanMode = "passive";

```

```

27
28     /**
29      * 是否允许缓存空值,默认:false
30      */
31     private boolean allowNullValues = false;
32
33     /**
34      * 使用哪种redis客户端,默认: jedis
35      * <ul>
36      * <li><a href ='https://github.com/xetorthio/jedis'>jedis:
https://github.com/xetorthio/jedis</a></li>
37      * <li><a href ='https://github.com/lettuce-io/lettuce-core'>lettuce:
https://github.com/lettuce-io/lettuce-core</a></li>
38      * </ul>
39      */
40     private String redisClient = "jedis";
41
42     /**
43      * 是否开启二级缓存
44      */
45     private boolean l2CacheOpen = true;
46
47
48     public String getConfigLocation()
49     {
50         return configLocation;
51     }
52
53     public void setConfigLocation(String configLocation)
54     {
55         this.configLocation = configLocation;
56     }
57
58     public Boolean getOpenSpringCache()
59     {
60         return openSpringCache;
61     }
62
63     public void setOpenSpringCache(Boolean openSpringCache)
64     {
65         this.openSpringCache = openSpringCache;
66     }
67
68     public String getCacheCleanMode()
69     {
70         return cacheCleanMode;
71     }
72
73     public void setCacheCleanMode(String cacheCleanMode)
74     {
75         this.cacheCleanMode = cacheCleanMode;
76     }
77
78     public boolean isAllowNullValues()
79     {
80         return allowNullValues;
81     }
82

```

```

83     public void setAllowNullValues(boolean allowNullValues)
84     {
85         this.allowNullValues = allowNullValues;
86     }
87
88     public String getRedisClient()
89     {
90         return redisClient;
91     }
92
93     public void setRedisClient(String redisClient)
94     {
95         this.redisClient = redisClient;
96     }
97
98     public boolean getL2CacheOpen()
99     {
100         return l2CacheOpen;
101     }
102
103     public void setL2CacheOpen(boolean l2CacheOpen)
104     {
105         this.l2CacheOpen = l2CacheOpen;
106     }
107
108 }

```

第十七步：添加配置类J2CacheSpringCacheAutoConfiguration

```

1  package net.oschina.j2cache.autoconfigure;
2
3  import net.oschina.j2cache.CacheChannel;
4  import net.oschina.j2cache.J2Cache;
5  import net.oschina.j2cache.cache.support.J2CacheCacheManger;
6  import org.springframework.boot.autoconfigure.cache.CacheProperties;
7  import org.springframework.boot.autoconfigure.condition.ConditionalOnBean;
8  import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
9  import
org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
10 import
org.springframework.boot.context.properties.EnableConfigurationProperties;
11 import org.springframework.cache.annotation.EnableCaching;
12 import org.springframework.context.annotation.Bean;
13 import org.springframework.context.annotation.Configuration;
14
15 import java.util.List;
16
17 /**
18  * 开启对spring cache支持的配置入口
19  */
20 @Configuration

```

```

21 @ConditionalOnClass(J2Cache.class)
22 @EnableConfigurationProperties({J2CacheConfig.class, CacheProperties.class})
23 @ConditionalOnProperty(name = "j2cache.open-spring-cache", havingValue =
    "true")
24 @EnableCaching
25 public class J2CacheSpringCacheAutoConfiguration
26 {
27
28     private final CacheProperties cacheProperties;
29
30     private final J2CacheConfig j2CacheConfig;
31
32     J2CacheSpringCacheAutoConfiguration(CacheProperties cacheProperties,
    J2CacheConfig j2CacheConfig)
33     {
34         this.cacheProperties = cacheProperties;
35         this.j2CacheConfig = j2CacheConfig;
36     }
37
38     @Bean
39     @ConditionalOnBean(CacheChannel.class)
40     public J2CacheCacheManger cacheManager(CacheChannel cacheChannel)
41     {
42         List<String> cacheNames = cacheProperties.getCacheNames();
43         J2CacheCacheManger cacheCacheManger = new
    J2CacheCacheManger(cacheChannel);
44
45         cacheCacheManger.setAllowNullValues(j2CacheConfig.isAllowNullValues());
46         cacheCacheManger.setCacheNames(cacheNames);
47         return cacheCacheManger;
48
49
50     }

```

第十八步：添加配置类J2CacheSpringRedisAutoConfiguration

```

1  package net.oschina.j2cache.autoconfigure;
2
3  import java.io.Serializable;
4  import java.net.URI;
5  import java.net.URISyntaxException;
6  import java.time.Duration;
7  import java.util.ArrayList;
8  import java.util.List;
9  import java.util.Properties;
10
11 import net.oschina.j2cache.cache.support.util.J2CacheSerializer;
12 import net.oschina.j2cache.redis.RedisUtils;
13 import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
14 import org.slf4j.Logger;

```

```

15 import org.slf4j.LoggerFactory;
16 import org.springframework.beans.factory.annotation.Qualifier;
17 import org.springframework.boot.autoconfigure.AutoConfigureAfter;
18 import org.springframework.boot.autoconfigure.AutoConfigureBefore;
19 import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
20 import
org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
21 import
org.springframework.boot.autoconfigure.data.redis.RedisAutoConfiguration;
22 import org.springframework.context.annotation.Bean;
23 import org.springframework.context.annotation.Configuration;
24 import org.springframework.context.annotation.Primary;
25 import org.springframework.data.redis.connection.RedisClusterConfiguration;
26 import org.springframework.data.redis.connection.RedisConnectionFactory;
27 import org.springframework.data.redis.connection.RedisNode;
28 import org.springframework.data.redis.connection.RedisPassword;
29 import
org.springframework.data.redis.connection.RedisSentinelConfiguration;
30 import
org.springframework.data.redis.connection.RedisStandaloneConfiguration;
31 import
org.springframework.data.redis.connection.jedis.JedisClientConfiguration;
32 import
org.springframework.data.redis.connection.jedis.JedisClientConfiguration.Je
disClientConfigurationBuilder;
33 import
org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
34 import
org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
35 import
org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfi
guration;
36 import
org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfi
guration.LettucePoolingClientConfigurationBuilder;
37 import org.springframework.data.redis.core.RedisTemplate;
38 import
org.springframework.data.redis.listener.RedisMessageListenerContainer;
39 import org.springframework.data.redis.serializer.RedisSerializer;
40 import org.springframework.data.redis.serializer.StringRedisSerializer;
41 import org.springframework.util.StringUtils;
42 import redis.clients.jedis.JedisPoolConfig;
43 import redis.clients.jedis.JedisShardInfo;
44 import redis.clients.jedis.exceptions.JedisConnectionException;
45
46 /**
47  * 对spring redis支持的配置入口
48  */
49 @Configuration
50 @AutoConfigureAfter({RedisAutoConfiguration.class})
51 @AutoConfigureBefore({J2CacheAutoConfiguration.class})
52 @ConditionalOnProperty(value = "j2cache.12-cache-open", havingValue =
"true", matchIfMissing = true)
53 public class J2CacheSpringRedisAutoConfiguration
54 {
55
56     private final static int MAX_ATTEMPTS = 3;

```

```

57
58     private final static int CONNECT_TIMEOUT = 5000;
59
60     private static final Logger log =
61         LoggerFactory.getLogger(net.oschina.j2cache.autoconfigure.J2CacheSpringRedi
62             sAutoConfiguration.class);
63
64         @SuppressWarnings("deprecation")
65         @Bean("j2CahceRedisConnectionFactory")
66         @ConditionalOnMissingBean(name = "j2CahceRedisConnectionFactory")
67         @ConditionalOnProperty(name = "j2cache.redis-client", havingValue =
68             "jedis", matchIfMissing = true)
69         public JedisConnectionFactory
70             jedisConnectionFactory(net.oschina.j2cache.J2CacheConfig j2CacheConfig)
71         {
72             Properties l2CacheProperties =
73                 j2CacheConfig.getL2CacheProperties();
74             String hosts = l2CacheProperties.getProperty("hosts");
75             String mode = l2CacheProperties.getProperty("mode") == null ?
76                 "null" : l2CacheProperties.getProperty("mode");
77             String clusterName = l2CacheProperties.getProperty("cluster_name");
78             String password = l2CacheProperties.getProperty("password");
79             int database = l2CacheProperties.getProperty("database") == null ?
80                 0
81                 :
82                 Integer.parseInt(l2CacheProperties.getProperty("database"));
83             JedisConnectionFactory connectionFactory = null;
84             JedisPoolConfig config =
85                 RedisUtils.newPoolConfig(l2CacheProperties, null);
86             List<RedisNode> nodes = new ArrayList<>();
87             if (hosts != null && !"".equals(hosts))
88             {
89                 for (String node : hosts.split(","))
90                 {
91                     String[] s = node.split(":");
92                     String host = s[0];
93                     int port = (s.length > 1) ? Integer.parseInt(s[1]) : 6379;
94                     RedisNode n = new RedisNode(host, port);
95                     nodes.add(n);
96                 }
97             }
98             else
99             {
100                 log.error("j2cache中的redis配置缺少hosts! !");
101                 throw new IllegalArgumentException("j2cache中的redis配置缺少
102                     hosts");
103             }
104
105             RedisPassword paw = RedisPassword.none();
106             if (!StringUtils.isEmpty(password))
107             {
108                 paw = RedisPassword.of(password);
109             }
110
111             switch (mode)
112             {
113                 case "sentinel":

```

```

104         RedisSentinelConfiguration sentinel = new
RedisSentinelConfiguration();
105         sentinel.setDatabase(database);
106         sentinel.setPassword(paw);
107         sentinel.setMaster(clusterName);
108         sentinel.setSentinels(nodes);
109         connectionFactory = new JedisConnectionFactory(sentinel,
config);
110         break;
111     case "cluster":
112         RedisClusterConfiguration cluster = new
RedisClusterConfiguration();
113         cluster.setClusterNodes(nodes);
114         cluster.setMaxRedirects(MAX_ATTEMPTS);
115         cluster.setPassword(paw);
116         connectionFactory = new JedisConnectionFactory(cluster,
config);
117         break;
118     case "sharded":
119         try
120         {
121             for (String node : hosts.split(","))
122             {
123                 connectionFactory = new JedisConnectionFactory(new
JedisShardInfo(new URI(node)));
124                 connectionFactory.setPoolConfig(config);
125                 log.warn("Jedis mode [sharded] not recommended for
use!!");
126                 break;
127             }
128         }
129         catch (URISyntaxException e)
130         {
131             throw new JedisConnectionException(e);
132         }
133         break;
134     default:
135         for (RedisNode node : nodes)
136         {
137             String host = node.getHost();
138             int port = node.getPort();
139             RedisStandaloneConfiguration single = new
RedisStandaloneConfiguration(host, port);
140             single.setDatabase(database);
141             single.setPassword(paw);
142             JedisClientConfigurationBuilder clientConfiguration =
JedisClientConfiguration.builder();
143             clientConfiguration.usePooling().poolConfig(config);
144
145             clientConfiguration.connectTimeout(Duration.ofMillis(CONNECT_TIMEOUT));
146             connectionFactory = new JedisConnectionFactory(single,
clientConfiguration.build());
147             break;
148         }
149         if (!"single".equalsIgnoreCase(mode))
150         {
151             log.warn("Redis mode [" + mode + "] not defined. Using
'single'.");

```

```

151         }
152         break;
153     }
154     return connectionFactory;
155 }
156
157 @Primary
158 @Bean("j2CahceRedisConnectionFactory")
159 @ConditionalOnMissingBean(name = "j2CahceRedisConnectionFactory")
160 @ConditionalOnProperty(name = "j2cache.redis-client", havingValue =
"lettuce")
161 public LettuceConnectionFactory
162 lettuceConnectionFactory(net.oschina.j2cache.J2CacheConfig j2CacheConfig)
163 {
164     Properties l2CacheProperties =
j2CacheConfig.getL2CacheProperties();
165     String hosts = l2CacheProperties.getProperty("hosts");
166     String mode = l2CacheProperties.getProperty("mode") == null ?
"null" : l2CacheProperties.getProperty("mode");
167     String clusterName = l2CacheProperties.getProperty("cluster_name");
168     String password = l2CacheProperties.getProperty("password");
169     int database = l2CacheProperties.getProperty("database") == null ?
0
:
Integer.parseInt(l2CacheProperties.getProperty("database"));
170     LettuceConnectionFactory connectionFactory = null;
171     LettucePoolingClientConfigurationBuilder config =
LettucePoolingClientConfiguration.builder();
172     config.commandTimeout(Duration.ofMillis(CONNECT_TIMEOUT));
173     config.poolConfig(getGenericRedisPool(l2CacheProperties, null));
174     List<RedisNode> nodes = new ArrayList<>();
175     if (hosts != null && !"".equals(hosts))
176     {
177         for (String node : hosts.split(","))
178         {
179             String[] s = node.split(":");
180             String host = s[0];
181             int port = (s.length > 1) ? Integer.parseInt(s[1]) : 6379;
182             RedisNode n = new RedisNode(host, port);
183             nodes.add(n);
184         }
185     }
186     else
187     {
188         log.error("j2cache中的redis配置缺少hosts! !");
189         throw new IllegalArgumentException();
190     }
191     RedisPassword paw = RedisPassword.none();
192     if (!StringUtils.isEmpty(password))
193     {
194         paw = RedisPassword.of(password);
195     }
196     switch (mode)
197     {
198         case "sentinel":
199             RedisSentinelConfiguration sentinel = new
RedisSentinelConfiguration();
200             sentinel.setDatabase(database);

```



```

201         sentinel.setPassword(paw);
202         sentinel.setMaster(clusterName);
203         sentinel.setSentinels(nodes);
204         connectionFactory = new LettuceConnectionFactory(sentinel,
config.build());
205         break;
206         case "cluster":
207             RedisClusterConfiguration cluster = new
RedisClusterConfiguration();
208             cluster.setClusterNodes(nodes);
209             cluster.setMaxRedirects(MAX_ATTEMPTS);
210             cluster.setPassword(paw);
211             connectionFactory = new LettuceConnectionFactory(cluster,
config.build());
212             break;
213             case "sharded":
214                 throw new IllegalArgumentException("Lettuce not support use
mode [sharded]!!");
215             default:
216                 for (RedisNode node : nodes)
217                 {
218                     String host = node.getHost();
219                     int port = node.getPort();
220                     RedisStandaloneConfiguration single = new
RedisStandaloneConfiguration(host, port);
221                     single.setDatabase(database);
222                     single.setPassword(paw);
223                     connectionFactory = new
LettuceConnectionFactory(single, config.build());
224                     break;
225                 }
226                 if (!"single".equalsIgnoreCase(mode))
227                 {
228                     log.warn("Redis mode [" + mode + "] not defined. Using
'single'.");
229                 }
230                 break;
231             }
232             return connectionFactory;
233         }
234
235         @Bean("j2CacheRedisTemplate")
236         public RedisTemplate<String, Serializable> j2CacheRedisTemplate(
237             @Qualifier("j2CahceRedisConnectionFactory")
RedisConnectionFactory j2CahceRedisConnectionFactory,
238             @Qualifier("j2CacheValueSerializer") RedisSerializer<Object>
j2CacheSerializer)
239         {
240             RedisTemplate<String, Serializable> template = new
RedisTemplate<String, Serializable>();
241             template.setKeySerializer(new StringRedisSerializer());
242             template.setHashKeySerializer(new StringRedisSerializer());
243             template.setDefaultSerializer(j2CacheSerializer);
244             template.setConnectionFactory(j2CahceRedisConnectionFactory);
245             return template;
246         }
247
248         @Bean("j2CacheValueSerializer")

```

```

249     @ConditionalOnMissingBean(name = "j2CacheValueSerializer")
250     public RedisSerializer<Object> j2CacheValueSerializer()
251     {
252         return new J2CacheSerializer();
253     }
254
255     @Bean("j2CacheRedisMessageListenerContainer")
256     RedisMessageListenerContainer container(
257         @Qualifier("j2CahceRedisConnectionFactory")
258         RedisConnectionFactory j2CahceRedisConnectionFactory)
259     {
260         RedisMessageListenerContainer container = new
261         RedisMessageListenerContainer();
262         container.setConnectionFactory(j2CahceRedisConnectionFactory);
263         return container;
264     }
265
266     private GenericObjectPoolConfig getGenericRedisPool(Properties props,
267     String prefix)
268     {
269         GenericObjectPoolConfig cfg = new GenericObjectPoolConfig();
270         cfg.setMaxTotal(Integer.valueOf((String)
271         props.getDefault(key(prefix, "maxTotal"), "-1")));
272         cfg.setMaxIdle(Integer.valueOf((String)
273         props.getDefault(key(prefix, "maxIdle"), "100")));
274         cfg.setMaxWaitMillis(Integer.valueOf((String)
275         props.getDefault(key(prefix, "maxWaitMillis"), "100")));
276         cfg.setMinEvictableIdleTimeMillis(
277         Integer.valueOf((String) props.getDefault(key(prefix,
278         "minEvictableIdleTimeMillis"), "864000000)));
279         cfg.setMinIdle(Integer.valueOf((String)
280         props.getDefault(key(prefix, "minIdle"), "10")));
281         cfg.setNumTestsPerEvictionRun(
282         Integer.valueOf((String) props.getDefault(key(prefix,
283         "numTestsPerEvictionRun"), "10")));
284         cfg.setLifo(Boolean.valueOf(props.getProperty(key(prefix, "lifo"),
285         "false")));
286         cfg.setSoftMinEvictableIdleTimeMillis(
287         Integer.valueOf((String) props.getDefault(key(prefix,
288         "softMinEvictableIdleTimeMillis"), "10")));
289         cfg.setTestOnBorrow(Boolean.valueOf(props.getProperty(key(prefix,
290         "testOnBorrow"), "true")));
291         cfg.setTestOnReturn(Boolean.valueOf(props.getProperty(key(prefix,
292         "testOnReturn"), "false")));
293         cfg.setTestWhileIdle(Boolean.valueOf(props.getProperty(key(prefix,
294         "testWhileIdle"), "true")));
295         cfg.setTimeBetweenEvictionRunsMillis(
296         Integer.valueOf((String) props.getDefault(key(prefix,
297         "timeBetweenEvictionRunsMillis"), "300000)));
298
299         cfg.setBlockWhenExhausted(Boolean.valueOf(props.getProperty(key(prefix,
300         "blockWhenExhausted"), "false")));
301         return cfg;
302     }
303
304     private String key(String prefix, String key)
305     {
306         return (prefix == null) ? key : prefix + "." + key;

```

```
290     }
291 }
```

第十九步：添加配置类CacheConfig

```
1  package mao.tools_j2cache.config;
2
3  import org.springframework.cache.annotation.CachingConfigurerSupport;
4  import org.springframework.cache.interceptor.KeyGenerator;
5
6  /**
7   * Project name(项目名称): j2cache_spring_boot_starter_demo
8   * Package(包名): mao.tools_j2cache.config
9   * Class(类名): CacheConfig
10  * Author(作者): mao
11  * Author QQ: 1296193245
12  * GitHub: https://github.com/maomao124/
13  * Date(创建日期): 2022/11/5
14  * Time(创建时间): 21:22
15  * Version(版本): 1.0
16  * Description(描述): 覆盖 SpringCache 相关配置
17  */
18
19  public class CacheConfig extends CachingConfigurerSupport
20  {
21      /**
22       * 解决注解: Cacheable 没有指定key时, 会将key生成 ${value}:SimpleKey []
23       * eg: @Cacheable(value = "pinda") -> pinda:SimpleKey []
24       *
25       * @return {@link KeyGenerator}
26       */
27      @Override
28      public KeyGenerator keyGenerator()
29      {
30          return (target, method, objects) ->
31          {
32              /*StringBuilder sb = new StringBuilder();
33              sb.append(target.getClass().getName());
34              sb.append(StrPool.COLON);
35              sb.append(method.getName());
36              for (Object obj : objects) {
37                  if (obj != null) {
38                      sb.append(StrPool.COLON);
39                      sb.append(obj.toString());
40                  }
41              }
42              return sb.toString();*/
43              return "";
44          };
45      }
```

第二十步：添加实体类RedisData

```

1  package mao.tools_j2cache.entity;
2
3  import java.time.LocalDateTime;
4
5  /**
6   * Project name(项目名称): j2cache_spring_boot_starter_demo
7   * Package(包名): mao.tools_j2cache.entity
8   * Class(类名): RedisData
9   * Author(作者): mao
10  * Author QQ: 1296193245
11  * GitHub: https://github.com/maomao124/
12  * Date(创建日期): 2022/11/5
13  * Time(创建时间): 22:29
14  * Version(版本): 1.0
15  * Description(描述): 无
16  */
17
18  public class RedisData
19  {
20      /**
21       * 数据
22       */
23      private Object data;
24
25      /**
26       * 过期时间
27       */
28      private LocalDateTime expireTime;
29
30      /**
31       * Instantiates a new Redis data.
32       */
33      public RedisData()
34      {
35
36      }
37
38      /**
39       * Instantiates a new Redis data.
40       *
41       * @param data the data
42       * @param expireTime the expire time
43       */
44      public RedisData(Object data, LocalDateTime expireTime)
45      {
46          this.data = data;

```

```

47         this.expireTime = expireTime;
48     }
49
50     /**
51      * Gets data.
52      *
53      * @return the data
54      */
55     public Object getData()
56     {
57         return data;
58     }
59
60     /**
61      * Sets data.
62      *
63      * @param data the data
64      */
65     public void setData(Object data)
66     {
67         this.data = data;
68     }
69
70     /**
71      * Gets expire time.
72      *
73      * @return the expire time
74      */
75     public LocalDateTime getExpireTime()
76     {
77         return expireTime;
78     }
79
80     /**
81      * Sets expire time.
82      *
83      * @param expireTime the expire time
84      */
85     public void setExpireTime(LocalDateTime expireTime)
86     {
87         this.expireTime = expireTime;
88     }
89
90     @Override
91     public String toString()
92     {
93         return "RedisData{" + "data=" + data +
94             ", expireTime=" + expireTime +
95             '}';
96     }
97 }

```

第二十一部：添加配置类RedissonConfig

```
1 package mao.tools_j2cache.config;
2
3 import org.redisson.Redisson;
4 import org.redisson.api.RedissonClient;
5 import org.redisson.config.ClusterServersConfig;
6 import org.redisson.config.Config;
7 import org.slf4j.Logger;
8 import org.slf4j.LoggerFactory;
9 import org.springframework.beans.factory.annotation.Value;
10 import org.springframework.context.annotation.Bean;
11 import org.springframework.context.annotation.Configuration;
12
13 import javax.annotation.PostConstruct;
14
15 /**
16  * Project name(项目名称): j2cache_spring_boot_starter_demo
17  * Package(包名): mao.tools_j2cache.config
18  * Class(类名): RedissonConfig
19  * Author(作者): mao
20  * Author QQ: 1296193245
21  * Github: https://github.com/maomao124/
22  * Date(创建日期): 2022/11/5
23  * Time(创建时间): 22:47
24  * Version(版本): 1.0
25  * Description(描述): 无
26  */
27
28 @Configuration
29 public class RedissonConfig
30 {
31     /**
32      * 日志
33      */
34     private static final Logger log =
35         LoggerFactory.getLogger(RedissonConfig.class);
36
37     /**
38      * Redisson配置
39      *
40      * @return RedissonClient
41      */
42     @Bean
43     public RedissonClient redissonClient(@Value("${redis.hosts}") String
44         hosts, @Value("${redis.password}") String password)
45     {
46         if (!hosts.contains(","))
47         {
48             //没有逗号，单机模式
49             log.info("单机模式redis:" + hosts);
50             //配置类
51             Config config = new Config();
52             //添加redis地址，用config.useClusterServers()添加集群地址
```

```

51         config.useSingleServer().setAddress("redis://" +
hosts).setPassword(password);
52         //创建客户端
53         return Redisson.create(config);
54     }
55     else
56     {
57         //集群
58         log.info("集群模式redis:" + hosts);
59         String[] hosts_ = hosts.split(",");
60         //配置类
61         Config config = new Config();
62         //添加redis地址，用config.useClusterServers()添加集群地址
63         ClusterServersConfig clusterServersConfig =
config.useClusterServers();
64         for (String host : hosts_)
65         {
66             clusterServersConfig.addNodeAddress("redis://" + host);
67         }
68         //config.useSingleServer().setAddress("redis://" +
hosts).setPassword(password);
69         //创建客户端
70         return Redisson.create(config);
71     }
72
73 }
74
75 @PostConstruct
76 public void init()
77 {
78     log.info("初始化 RedissonConfig");
79 }
80 }

```

第二十二步：添加工具类RedisUtils

```

1  package mao.tools_j2cache.utils;
2
3  import cn.hutool.core.util.StrUtil;
4  import cn.hutool.json.JSONObject;
5  import cn.hutool.json.JSONUtil;
6  import mao.tools_j2cache.entity.RedisData;
7  import org.redisson.api.RLock;
8  import org.redisson.api.RedissonClient;
9  import org.slf4j.Logger;
10 import org.slf4j.LoggerFactory;
11 import org.springframework.data.redis.core.StringRedisTemplate;
12
13
14 import javax.annotation.Resource;
15 import java.time.LocalDateTime;

```

```

16 import java.util.concurrent.ExecutorService;
17 import java.util.concurrent.Executors;
18 import java.util.concurrent.TimeUnit;
19 import java.util.function.Function;
20
21
22 /**
23  * Project name(项目名称): j2cache_spring_boot_starter_demo
24  * Package(包名): mao.tools_j2cache.utils
25  * Class(类名): RedisUtils
26  * Author(作者): mao
27  * Author QQ: 1296193245
28  * Github: https://github.com/maomao124/
29  * Date(创建日期): 2022/11/5
30  * Time(创建时间): 22:25
31  * Version(版本): 1.0
32  * Description(描述): 缓存工具类
33  */
34
35
36 public class RedisUtils
37 {
38
39     /**
40      * 日志
41      */
42     private static final Logger log =
43         LoggerFactory.getLogger(RedisUtils.class);
44
45     @Resource
46     private StringRedisTemplate stringRedisTemplate;
47
48     @Resource
49     private RedissonClient redissonClient;
50
51     //线程池
52     private static final ExecutorService CACHE_REBUILD_EXECUTOR =
53         Executors.newFixedThreadPool(10);
54
55     /**
56      * 向redis里添加数据
57      *
58      * @param redisKey    redis的key
59      * @param value        数据
60      * @param expireTime 过期时间
61      * @param timeUnit    时间单位
62      */
63     public void set(String redisKey, Object value, Long expireTime,
64                     TimeUnit timeUnit)
65     {
66         stringRedisTemplate.opsForValue().set(redisKey,
67             JSONUtil.toJsonStr(value), expireTime, timeUnit);
68     }
69
70     /**
71      * 向redis里添加数据 设置逻辑过期
72      *

```



```

70     * @param redisKey    redis的key
71     * @param value      数据
72     * @param expireTime 过期时间
73     * @param timeUnit   时间单位
74     */
75     public void setWithLogicalExpire(String redisKey, Object value, Long
expireTime, TimeUnit timeUnit)
76     {
77         RedisData redisData = new RedisData();
78         //添加数据
79         redisData.setData(value);
80         //设置过期时间
81
82         redisData.setExpireTime(LocalDateTime.now().plusSeconds(timeUnit.toSeconds
(expireTime)));
83         //放入redis
84         stringRedisTemplate.opsForValue().set(redisKey,
JSONUtil.toJsonStr(redisData));
85     }
86
87     /**
88     * 查询数据，有缓存，解决缓存穿透问题，未解决缓存雪崩问题
89     *
90     * @param <R>          返回值的类型
91     * @param <ID>         id的类型
92     * @param keyPrefix    redisKey的前缀
93     * @param id           id
94     * @param type         返回值的类型
95     * @param dbFallback   查询数据库的函数
96     * @param expireTime   过期时间
97     * @param timeUnit     时间单位
98     * @return 泛型R r
99     */
100    public <R, ID> R queryWithPassThrough(String keyPrefix, ID id, Class<R>
type,
101                                           Function<ID, R> dbFallback, Long
expireTime, TimeUnit timeUnit)
102    {
103        //获得前缀
104        String redisKey = keyPrefix + id;
105        //查询redis
106        String json = stringRedisTemplate.opsForValue().get(redisKey);
107        //判断是否为空
108        if (StrUtil.isNotBlank(json))
109        {
110            //不为空，返回
111            return JSONUtil.toBean(json, type);
112        }
113        //判断是否为空串
114        if (json != null)
115        {
116            //空串
117            return null;
118        }
119        //null
120        //查数据库
121        R r = dbFallback.apply(id);

```

```

122         //判断
123         if (r == null)
124         {
125             //数据库也为空，缓存空值
126             this.set(redisKey, "", expireTime, timeUnit);
127             return null;
128         }
129         //数据库存在，写入redis
130         this.set(redisKey, r, expireTime, timeUnit);
131         //返回
132         return r;
133     }
134
135     /**
136      * 查询数据，有缓存，解决缓存穿透问题，解决缓存雪崩问题
137      *
138      * @param <R>          返回值的类型
139      * @param <ID>         id的类型
140      * @param keyPrefix    redisKey的前缀
141      * @param id           id
142      * @param type         返回值的类型
143      * @param dbFallback   查询数据库的函数
144      * @param expireTime   过期时间
145      * @param timeUnit     时间单位
146      * @param maxTimeSecondsByCacheAvalanche this.set(redisKey, r,
147      *
148      *                               TimeUnit.SECONDS);
149      * @return 泛型R r
150      */
151     public <R, ID> R queryWithPassThroughAndCacheAvalanche(String
keyPrefix, ID id, Class<R> type,
152                                                         Function<ID, R>
dbFallback, Long expireTime, TimeUnit timeUnit,
153                                                         Integer
maxTimeSecondsByCacheAvalanche)
154     {
155         //获得前缀
156         String redisKey = keyPrefix + id;
157         //查询redis
158         String json = stringRedisTemplate.opsForValue().get(redisKey);
159         //判断是否为空
160         if (StrUtil.isNotBlank(json))
161         {
162             //不为空，返回
163             return JSONUtil.toBean(json, type);
164         }
165         //判断是否为空串
166         if (json != null)
167         {
168             //空串
169             return null;
170         }
171         //null
172         //查数据库
173         R r = dbFallback.apply(id);
174         //判断

```

```

175         if (r == null)
176         {
177             //数据库也为空，缓存空值
178             this.set(redisKey, "",
179                 timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
180                 TimeUnit.SECONDS);
181             return null;
182         }
183         //数据库存在，写入redis
184         this.set(redisKey, r,
185             timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
186             TimeUnit.SECONDS);
187         //返回
188         return r;
189     }
190
191     /**
192     * 查询数据，解决缓存穿透，互斥锁方法解决缓存击穿，解决缓存雪崩
193     *
194     * @param <R>                返回值的类型
195     * @param <ID>               id的类型
196     * @param keyPrefix          redisKey的前缀
197     * @param lockKeyPrefix      锁的前缀
198     * @param id                 id
199     * @param type               返回值的类型
200     * @param dbFallback         查询数据库的函数
201     * @param expireTime         过期时间
202     * @param timeUnit           时间单位
203     * @param maxTimeSecondsByCacheAvalanche this.set(redisKey, r,
204     *
timeUnit.toSeconds(expireTime)+getIntRandom(0,maxTimeSecondsByCacheAvalanch
e),
205     * @return 泛型R r
206     */
207     public <R, ID> R query(String keyPrefix, String lockKeyPrefix, ID id,
Class<R> type,
208                           Function<ID, R> dbFallback, Long expireTime,
TimeUnit timeUnit,
209                           Integer maxTimeSecondsByCacheAvalanche)
210     {
211         //获取redisKey
212         String redisKey = keyPrefix + id;
213         //log.debug("查询: " + redisKey);
214         //从redis中查询信息，根据id
215         String json = stringRedisTemplate.opsForValue().get(redisKey);
216         //判断取出的数据是否为空
217         if (StrUtil.isNotBlank(json))
218         {
219             //log.debug(redisKey + " 缓存命中");
220             //不是空，redis里有，返回
221             return JSONUtil.toBean(json, type);
222         }
223         //是空串，不是null，返回
224         if (json != null)
225         {
226             return null;

```

```

227     }
228     //锁的key
229     String lockKey = lockKeyPrefix + id;
230
231     R r = null;
232     LockInfo lockInfo = null;
233     try
234     {
235         //log.debug(redisKey + " 缓存未命中，尝试获取锁");
236         //获取互斥锁
237         lockInfo = tryLock(lockKey);
238         //判断锁是否获取成功
239         if (!lockInfo.isSuccess())
240         {
241             //没有获取到锁
242             //200毫秒后再次获取
243             Thread.sleep(200);
244             //递归调用
245             return query(keyPrefix, lockKeyPrefix, id, type,
dbFallback,
246
247                 expireTime, timeUnit,
maxTimeSecondsByCacheAvalanche);
248         }
249         //得到了锁
250         //从redis中查询信息，根据id
251         json = stringRedisTemplate.opsForValue().get(redisKey);
252         //判断取出的数据是否为空
253         if (StrUtil.isNotBlank(json))
254         {
255             //log.debug(redisKey + " 获取分布式锁后，缓存命中");
256             //不是空，redis里有，返回
257             return JSONUtil.toBean(json, type);
258         }
259
260         //null，查数据库
261         log.debug(redisKey + " 获取分布式锁后，缓存未命中，查询数据库");
262         r = dbFallback.apply(id);
263         //判断数据库里的信息是否为空
264         if (r == null)
265         {
266             //数据库也为空，缓存空值
267             this.set(redisKey, "",
268                 timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
269                 TimeUnit.SECONDS);
270             return null;
271         }
272         //存在，回写到redis里，设置随机的过期时间
273         this.set(redisKey, r,
274             timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
275             TimeUnit.SECONDS);
276     }
277     catch (InterruptedException e)
278     {
279         throw new RuntimeException(e);
280     }
281     finally

```

```

281         {
282             //释放锁
283             if (lockInfo != null)
284             {
285                 this.unlock(lockInfo);
286             }
287         }
288         //返回数据
289         return r;
290     }
291
292     /**
293     * 更新数据
294     *
295     * @param <T>          要更新的对象的泛型
296     * @param <ID>         主键的类型
297     * @param id           要更新的主键
298     * @param data         要更新的对象
299     * @param keyPrefix    redis的key前缀
300     * @param dbFallback 更新数据库的函数，返回值要为Boolean类型
301     * @return boolean boolean
302     */
303     public <T, ID> boolean update(ID id, T data, String keyPrefix,
304     Function<T, Boolean> dbFallback)
305     {
306         //判断是否为空
307         if (id == null)
308         {
309             return false;
310         }
311         //不为空
312         //先更新数据库
313         boolean b = dbFallback.apply(data);
314         //更新失败，返回
315         if (!b)
316         {
317             return false;
318         }
319         //更新没有失败
320         //删除redis里的数据，下一次查询时自动添加进redis
321         //redisKey
322         String redisKey = keyPrefix + id;
323         stringRedisTemplate.delete(redisKey);
324         log.debug("更新: " + redisKey);
325         //返回响应
326         return true;
327     }
328
329     /**
330     * Query with logical expire r.
331     *
332     * @param <R>          返回值的类型
333     * @param <ID>         id的类型
334     * @param keyPrefix    redisKey的前缀
335     * @param lockKeyPrefix 锁的前缀
336     * @param id           id
337     * @param type         返回值的类型
338     * @param dbFallback   查询数据库的函数

```

```

338     * @param time          过期时间
339     * @param timeUnit      时间单位
340     * @return 泛型R r
341     */
342     public <R, ID> R queryWithLogicalExpire(String keyPrefix, String
lockKeyPrefix, ID id, Class<R> type,
343                                         Function<ID, R> dbFallback,
Long time, TimeUnit timeUnit)
344     {
345         //获得前缀
346         String redisKey = keyPrefix + id;
347         //查询redis
348         String json = stringRedisTemplate.opsForValue().get(redisKey);
349         //判断是否为空
350         if (StrUtil.isBlank(json))
351         {
352             //空, 返回
353             return null;
354         }
355         //不为空
356         //json 反序列化为对象
357         RedisData redisData = JSONUtil.toBean(json, RedisData.class);
358         //获得过期时间
359         LocalDateTime expireTime = redisData.getExpireTime();
360         //获取数据
361         R r = JSONUtil.toBean((JSONObject) redisData.getData(), type);
362         //判断是否过期
363         if (expireTime.isAfter(LocalDateTime.now()))
364         {
365             //未过期, 返回
366             return r;
367         }
368         //过期, 缓存重建
369         //获取互斥锁
370         String lockKey = lockKeyPrefix + id;
371         LockInfo lockInfo = tryLock(lockKey);
372         if (lockInfo.isSuccess())
373         {
374             //获取锁成功
375             // 开辟独立线程
376             CACHE_REBUILD_EXECUTOR.submit(new Runnable()
377             {
378                 @Override
379                 public void run()
380                 {
381                     try
382                     {
383                         R r1 = dbFallback.apply(id);
384                         setWithLogicalExpire(redisKey, r1, time, timeUnit);
385                     }
386                     catch (Exception e)
387                     {
388                         throw new RuntimeException(e);
389                     }
390                     finally
391                     {
392                         //释放锁
393                         unlock(lockInfo);

```

```

394         }
395     }
396     });
397 }
398 //没有获取到锁，使用旧数据返回
399 return r;
400 }
401
402
403 /**
404  * 获取锁
405  *
406  * @param key redisKey
407  * @return {@link LockInfo}
408  */
409 private LockInfo tryLock(String key)
410 {
411     // 获取锁（可重入），指定锁的名称
412     RLock lock = redissonClient.getLock(key);
413     try
414     {
415         // 尝试获取锁，参数分别是：获取锁的最大等待时间（期间会重试），锁自动释放时
416         // 间，时间单位
417         LockInfo lockInfo = new LockInfo();
418         lockInfo.setLock(lock);
419         lockInfo.setSuccess(lock.tryLock(1, 10, TimeUnit.SECONDS));
420         //log.debug("尝试获取分布式锁: " + lockInfo.isSuccess());
421         return lockInfo;
422     }
423     catch (InterruptedException e)
424     {
425         e.printStackTrace();
426         LockInfo lockInfo = new LockInfo();
427         lockInfo.setLock(lock);
428         lockInfo.setSuccess(false);
429         return lockInfo;
430     }
431 }
432
433 /**
434  * 释放锁
435  *
436  * @param lockInfo 锁信息
437  */
438 private void unlock(LockInfo lockInfo)
439 {
440     if (lockInfo.getLock().isHeldByCurrentThread())
441     {
442         //log.debug("尝试释放分布式锁");
443         lockInfo.lock.unlock();
444     }
445 }
446
447 /**
448  * 获取一个随机数，区间包含min和max
449  *
450  * @param min 最小值

```

```

451     * @param max 最大值
452     * @return int 型的随机数
453     */
454     @SuppressWarnings("all")
455     private int getIntRandom(int min, int max)
456     {
457         if (min > max)
458         {
459             min = max;
460         }
461         return min + (int) (Math.random() * (max - min + 1));
462     }
463
464     private static class LockInfo
465     {
466         /**
467          * 获取锁是否成功
468          */
469         private boolean isSuccess;
470
471         /**
472          * 锁对象
473          */
474         private RLock lock;
475
476         /**
477          * Is success boolean.
478          *
479          * @return the boolean
480          */
481         public boolean isSuccess()
482         {
483             return isSuccess;
484         }
485
486         /**
487          * Sets success.
488          *
489          * @param success the success
490          */
491         public void setSuccess(boolean success)
492         {
493             isSuccess = success;
494         }
495
496         /**
497          * Gets lock.
498          *
499          * @return the lock
500          */
501         public RLock getLock()
502         {
503             return lock;
504         }
505
506         /**
507          * Sets lock.
508          *

```



```

509         * @param lock the lock
510         */
511         public void setLock(RLock lock)
512         {
513             this.lock = lock;
514         }
515     }
516 }

```

第二十三步：添加配置类RedisUtilsConfig

```

1  package mao.tools_j2cache.config;
2
3  import mao.tools_j2cache.utils.RedisUtils;
4  import org.slf4j.Logger;
5  import org.slf4j.LoggerFactory;
6  import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
7  import org.springframework.context.annotation.Bean;
8  import org.springframework.context.annotation.Configuration;
9  import org.springframework.context.annotation.Import;
10
11  import javax.annotation.PostConstruct;
12
13  /**
14   * Project name(项目名称): j2cache_spring_boot_starter_demo
15   * Package(包名): mao.tools_j2cache.config
16   * Class(类名): RedisUtilsConfig
17   * Author(作者): mao
18   * Author QQ: 1296193245
19   * Github: https://github.com/maomao124/
20   * Date(创建日期): 2022/11/5
21   * Time(创建时间): 23:30
22   * Version(版本): 1.0
23   * Description(描述): 无
24   */
25
26  @Configuration
27  @ConditionalOnClass(RedisUtils.class)
28  @Import(RedisUtils.class)
29  public class RedisUtilsConfig
30  {
31      /**
32       * 日志
33       */
34      private static final Logger log =
35          LoggerFactory.getLogger(RedisUtilsConfig.class);
36
37      @PostConstruct
38      public void init()
39      {
40          log.info("初始化 RedisUtilsConfig");
41      }
42  }

```

```
40     }
41 }
```

第二十四步：编写spring.factories

```
1 org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
2     mao.tools_j2cache.config.CacheConfig,\
3     net.oschina.j2cache.autoconfigure.J2CacheAutoConfiguration,\
4     net.oschina.j2cache.autoconfigure.J2CacheSpringCacheAutoConfiguration,\
5     net.oschina.j2cache.autoconfigure.J2CacheSpringRedisAutoConfiguration,\
6     mao.tools_j2cache.config.RedissonConfig,\
7     mao.tools_j2cache.config.RedisUtilsConfig
```

使用starter

第一步：添加tools-j2cache的依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <parent>
8         <artifactId>j2cache_spring_boot_starter_demo</artifactId>
9         <groupId>mao</groupId>
10        <version>0.0.1-SNAPSHOT</version>
11    </parent>
12    <artifactId>use-starter</artifactId>
13    <name>use-starter</name>
14    <description>use-starter</description>
15
16    <properties>
17
18    </properties>
```

```

17
18     <dependencies>
19
20         <dependency>
21             <groupId>org.springframework.boot</groupId>
22             <artifactId>spring-boot-starter-web</artifactId>
23         </dependency>
24
25         <dependency>
26             <groupId>org.springframework.boot</groupId>
27             <artifactId>spring-boot-starter-test</artifactId>
28             <scope>test</scope>
29         </dependency>
30
31         <dependency>
32             <groupId>mao</groupId>
33             <artifactId>tools-j2cache</artifactId>
34             <version>0.0.1-SNAPSHOT</version>
35         </dependency>
36
37     </dependencies>
38
39     <build>
40         <plugins>
41             <plugin>
42                 <groupId>org.springframework.boot</groupId>
43                 <artifactId>spring-boot-maven-plugin</artifactId>
44             </plugin>
45         </plugins>
46     </build>
47
48 </project>

```

第二步：编写配置文件application.yml

```

1  # 该配置文件，只做注释参考，请勿修改此文件。（修改后也不会有任何效果，如要修改配置，请在
   # nacos中修改redis.yml）
2
3  def:
4      redis:
5          ip: 127.0.0.1
6          port: 6379
7          password: 123456
8
9  # redis 通用配置
10 spring:
11     cache:
12         type: GENERIC
13
14 j2cache:

```

```

15 # j2cache 配置文件的在项目中/resources文件夹下的路径 （注意，若想将里面的配置存放在
naocs或者application.yml中，需要注释这行）
16 # configLocation: /j2cache.properties
17 # 是否开启 SpringCache 支持
18 open-spring-cache: true
19 # 清除缓存的模式
20 # active:主动清除，二级缓存过期主动通知各节点清除，优点在于所有节点可以同时收到缓存清
除
21 # passive:被动清除，一级缓存过期进行通知各节点清除一二级缓存
22 # blend:两种模式一起运作，对于各个节点缓存准确以及及时性要求高的可以使用，正常用前两
种模式中一个就可
23 cache-clean-mode: passive
24 # 是否允许存放null 值
25 allow-null-values: true
26 # redis 客户端 （可选值: jedis lettuce)
27 redis-client: lettuce
28 # 是否开启二级缓存 开发环境可以关闭
29 l2-cache-open: true
30
31 #以下来自 j2cache.properties
32 # 缓存广播方式:
33 # jgroups -> 使用jgroups的多播
34 # redis -> 使用redis发布/订阅机制(使用jedis)
35 # lettuce -> 使用redis发布/订阅机制(使用lettuce, 推荐)
36 # rabbitmq -> 使用 RabbitMQ 发布/消费 机制
37 # rocketmq -> 使用 RocketMQ 发布/消费 机制
38 # none -> 不要通知集群中的其他节点
39 # xx.xxxx.xxxx.Xxxxx 实现net.oschina.j2cache.Cluster.ClusterPolicy的您自己
的缓存广播策略类名
40 broadcast:
net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
41 # 1级缓存提供商类，可选值:
42 # none -> 禁用此级别缓存
43 # ehcache -> 使用 ehcache2 作为1级缓存
44 # ehcache3 -> 使用 ehcache3 作为1级缓存
45 # caffeine -> 使用 caffeine 作为1级缓存 （仅作用于内存）
46 L1:
47 provider_class: caffeine
48 # 2级缓存提供商类，可选值:
49 # redis -> 使用 redis 作为2级缓存 （使用 jedis)
50 # lettuce -> 使用 redis 作为2级缓存 （使用 lettuce)
51 # readonly-redis -> 使用 作为2级缓存 ,但永远不要向它写入数据。如果使用此提供程序，
则必须取消注释' j2cache.L2.config_section '使redis配置可用。
52 # memcached -> 使用 memcached 作为2级缓存 （使用 xmemcached),
53 # [classname] -> 使用自定义供应商 （当使用自定义时，必须手动指定
L2.config_section )
54 L2:
55 provider_class:
net.oschina.j2cache.cache.support.redis.SpringRedisProvider
56 config_section: lettuce
57 # 在redis缓存数据中启用/禁用ttl(如果禁用，redis中的对象将永远不会过期，默认值为
true)
58 # 注意:redis哈希模式(redis.storage = hash 和 lettuce.storage = hash)不支持此
功能
59 sync_ttl_to_redis: true
60 # 是否默认缓存空对象(默认为false)
61 default_cache_null_object: false
62 # 缓存序列化提供者 ， 可选值:

```

63	# fst -> 使用 fast 序列化 (推荐)	缺点: 增删改字段后反序列化会报错
64	# kyro -> 使用 kyro 序列化	缺点: 生成的byte数据中部包含field
	数据, 对类升级的兼容性很差。 跨语言支持较复杂!	
65	# json -> 使用 fst's json 序列化 (测试中)	缺点: 不支持LocalDateTime
66	# fastjson -> 使用 fastjson 序列化	缺点: 嵌入非静态类不支持, 阿里的东西bug多...
67	# java -> java 标准序列化	缺点: 速度慢, 占空间, 增删改字段后反序列化会报错
68	# xxx.xxx.xxxx.Xxx -> [自定义序列化类]	
69	serialization: json	
70		
71	# j2cache.serialization=json 时可用	
72	#json:	
73	# map.person: net.oschina.j2cache.demo.Person	
74		
75	# 广播相关配置: jgroups 配置 (当 j2cache.broadcast=jgroups 时, 才需要配置)	
76	jgroups:	
77	# 网络配置文件路径 (相对于/resources 目录)	
78	configXml: /network.xml	
79	# 广播渠道名称	
80	channel:	
81	name: j2cache	
82		
83	# 广播相关配置: rabbitmq 配置 (当 j2cache.broadcast=rabbitmq 时, 才需要配置)	
84	rabbitmq:	
85	exchange: j2cache	
86	host: localhost	
87	port: 5672	
88	username: guest	
89	password: guest	
90		
91	# 广播相关配置: rocketmq 配置 (当 j2cache.broadcast=rocketmq 时, 才需要配置)	
92	rocketmq:	
93	name: j2cache	
94	topic: j2cache	
95	# 使用;分割多台主机	
96	hosts: 127.0.0.1:9876	
97		
98	# 1级相关缓存配置: (当 j2cache.L1.config_section=ehcache 时, 才需要配置)	
99	ehcache:	
100	configXml: /ehcache.xml	
101		
102	# 1级相关缓存配置: (当 j2cache.L1.config_section=ehcache3 时, 才需要配置)	
103	ehcache3:	
104	configXml: /ehcache.xml	
105	defaultHeapSize: 1000	
106		
107	# 1级相关缓存配置: (当 j2cache.L1.config_section=caffeine 时, 才需要配置)	
108	caffeine:	
109	# properties 和 region.[name] 任选一种方式配置	
110	properties: /j2cache/caffeine.properties # 这个配置文件需要放在项目中	
111	#region.[name]: size, xxxx[s m h d]	
112		
113		
114	# 广播相关配置: redis 配置 (当 j2cache.broadcast=redis 时 或者 j2cache.L2.config_section=redis 时, 才需要配置)	

```

115 # 2级缓存相关配置: redis 配置 (当 j2cache.broadcast=redis 时 或者
j2cache.L2.config_section=redis 或者 j2cache.L2.provider_class=redis 时, 才需
要配置)
116 redis:
117     # Redis 集群模式
118     # single -> 单 redis 服务
119     # sentinel -> 主从 服务
120     # cluster -> 集群 服务 (数据库配置无效, 使用 database = 0)
121     # sharded -> 分片 服务 (密码、数据库必须在 hosts 中指定, 且连接池配置无效 ;
redis://user:password@127.0.0.1:6379/0)
122     mode: single
123     # redis storage mode (generic|hash)
124     storage: generic
125     # redis发布/订阅频道名称
126     channel: j2cache
127     # redis发布/订阅服务器(该值为空时, 使用redis.host)
128     channel.host:
129     # 集群名: 仅用于分片
130     cluster_name: j2cache
131     # redis缓存命名空间可选, 默认[空]
132     namespace:
133     hosts: ${def.redis.ip}:${def.redis.port}
134     timeout: 2000
135     password: ${def.redis.password}
136     database: 0
137     maxTotal: 100
138     maxIdle: 10
139     maxWaitMillis: 5000
140     minEvictableIdleTimeMillis: 60000
141     minIdle: 1
142     numTestsPerEvictionRun: 10
143     lifo: false
144     softMinEvictableIdleTimeMillis: 10
145     testOnBorrow: true
146     testOnReturn: false
147     testWhileIdle: true
148     timeBetweenEvictionRunsMillis: 300000
149     blockWhenExhausted: false
150     jmxEnabled: false
151
152 # 广播相关配置: lettuce 配置 (当 j2cache.broadcast=lettuce 或者
j2cache.L2.config_section=lettuce 时, 才需要配置)
153 # 2级缓存相关配置: lettuce 配置 (当 j2cache.broadcast=lettuce 或者
j2cache.L2.config_section=lettuce 或者 j2cache.L2.provider_class=redis 时,
才需要配置)
154 lettuce:
155     mode: single
156     namespace:
157     storage: generic
158     channel: j2cache
159     scheme: redis
160     hosts: ${def.redis.ip}:${def.redis.port}
161     password: ${def.redis.password}
162     database: 0
163     sentinelMasterId:
164     maxTotal: 100
165     maxIdle: 10
166     minIdle: 10

```

```

167     timeout: 10000
168
169     # 广播相关配置: memcached 配置      (当 j2cache.broadcast=memcached 或者
j2cache.L2.config_section=memcached 时, 才需要配置)
170     # 2级缓存相关配置: memcached 配置 (当 j2cache.broadcast=memcached 或者
j2cache.L2.config_section=memcached 或者
j2cache.L2.provider_class=memcached 时, 才需要配置)
171     memcached:
172         servers: 127.0.0.1:11211
173         username:
174         password:
175         connectionPoolSize: 10
176         connectTimeout: 1000
177         failureMode: false
178         healSessionInterval: 1000
179         maxQueuedNoReplyOperations: 100
180         opTimeout: 100
181         sanitizeKeys: false
182
183
184
185
186
187
188     # 设置日志级别, root表示根节点, 即整体应用日志级别
189     logging:
190         # 日志输出到文件的文件名
191         file:
192             name: server.log
193         # 字符集
194         charset:
195             file: UTF-8
196         # 分文件
197         logback:
198             rollingpolicy:
199                 #最大文件大小
200                 max-file-size: 16KB
201                 # 文件格式
202                 file-name-pattern: logs/server_log/%d{yyyy/MM月/dd日/}%i.log
203         # 设置日志组
204         group:
205             # 自定义组名, 设置当前组中所包含的包
206             mao_pro: mao
207         level:
208             root: info
209             # 为对应组设置日志级别
210             mao_pro: debug
211             # 日志输出格式
212         # pattern:
213         # console: "%d %clr(%p) --- [%16t] %clr(%-40.40c){cyan} : %m %n"

```

第三步：编写实体类Student

```
1 package mao.use_starter.entity;
2
3 /**
4  * Project name(项目名称): j2cache_spring_boot_starter_demo
5  * Package(包名): mao.use_starter.entity
6  * Class(类名): Student
7  * Author(作者): mao
8  * Author QQ: 1296193245
9  * GitHub: https://github.com/maomao124/
10 * Date(创建日期): 2022/11/5
11 * Time(创建时间): 23:51
12 * Version(版本): 1.0
13 * Description(描述): 无
14 */
15
16
17 public class Student
18 {
19     private Long id;
20     private String name;
21
22     /**
23      * Instantiates a new Student.
24      */
25     public Student()
26     {
27
28     }
29
30     /**
31      * Instantiates a new Student.
32      *
33      * @param id the id
34      * @param name the name
35      */
36     public Student(Long id, String name)
37     {
38         this.id = id;
39         this.name = name;
40     }
41
42     /**
43      * Gets id.
44      *
45      * @return the id
46      */
47     public Long getId()
48     {
49         return id;
50     }
51
52     /**
53      * Sets id.
```



```

54     *
55     * @param id the id
56     */
57     public void setId(Long id)
58     {
59         this.id = id;
60     }
61
62     /**
63     * Gets name.
64     *
65     * @return the name
66     */
67     public String getName()
68     {
69         return name;
70     }
71
72     /**
73     * Sets name.
74     *
75     * @param name the name
76     */
77     public void setName(String name)
78     {
79         this.name = name;
80     }
81 }

```

第四步：编写TestController

```

1  package mao.use_starter.controller;
2
3  import mao.tools_j2cache.utils.RedisUtils;
4  import mao.use_starter.entity.Student;
5  import net.oschina.j2cache.CacheChannel;
6  import net.oschina.j2cache.CacheObject;
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.web.bind.annotation.GetMapping;
11 import org.springframework.web.bind.annotation.RestController;
12
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.concurrent.TimeUnit;
16 import java.util.function.Function;
17
18 /**
19  * Project name(项目名称): j2cache_demo
20  * Package(包名): mao.j2cache_demo.controller

```

```

21  * Class(类名): TestController
22  * Author(作者): mao
23  * Author QQ: 1296193245
24  * GitHub: https://github.com/maomao124/
25  * Date(创建日期): 2022/11/5
26  * Time(创建时间): 13:22
27  * Version(版本): 1.0
28  * Description(描述): 无
29  */
30
31  @RestController
32  public class TestController
33  {
34
35      private static final Logger log =
36      LoggerFactory.getLogger(TestController.class);
37
38      @Autowired
39      private CacheChannel cacheChannel;
40
41      private final String key = "myKey";
42      private final String region = "rx";
43
44      @GetMapping("/getInfos")
45      public List<String> getInfos()
46      {
47          CacheObject cacheObject = cacheChannel.get(region, key);
48          if (cacheObject.getValue() == null)
49          {
50              log.info("查询数据库");
51              //缓存中没有找到, 查询数据库获得
52              List<String> data = new ArrayList<>();
53              data.add("info1");
54              data.add("info2");
55              try
56              {
57                  Thread.sleep(9);
58              }
59              catch (InterruptedException e)
60              {
61                  e.printStackTrace();
62              }
63              //放入缓存
64              cacheChannel.set(region, key, data);
65              return data;
66          }
67          return (List<String>) cacheObject.getValue();
68      }
69
70      private String cache = null;
71
72      @GetMapping("/getInfos2")
73      public String getInfos2()
74      {
75          if (cache == null)
76          {
77              log.info("查询数据库2");

```

```

78         try
79         {
80             Thread.sleep(10);
81         }
82         catch (InterruptedException e)
83         {
84             e.printStackTrace();
85         }
86         cache = "hello";
87     }
88     else
89     {
90         return cache;
91     }
92     return cache;
93 }
94
95
96 @GetMapping("/getInfos3")
97 public List<String> getInfos3()
98 {
99     CacheObject cacheObject = cacheChannel.get(region, key);
100     if (cacheObject.getValue() == null)
101     {
102         log.info("查询数据库3");
103         //缓存中没有找到，查询数据库获得
104         try
105         {
106             Thread.sleep(9);
107         }
108         catch (InterruptedException e)
109         {
110             e.printStackTrace();
111         }
112         //放入缓存
113         cacheChannel.set(region, key, null);
114         return null;
115     }
116     return null;
117 }
118
119 /**
120  * 清理指定缓存
121  *
122  * @return {@link String}
123  */
124 @GetMapping("/evict")
125 public String evict()
126 {
127     cacheChannel.evict(region, key);
128     return "evict success";
129 }
130
131 /**
132  * 检测存在哪级缓存
133  *
134  * @return {@link String}
135  */

```

```

136     @GetMapping("/check")
137     public String check()
138     {
139         int check = cacheChannel.check(region, key);
140         return "level:" + check;
141     }
142
143     /**
144      * 检测缓存数据是否存在
145      *
146      * @return {@link String}
147      */
148     @GetMapping("/exists")
149     public String exists()
150     {
151         boolean exists = cacheChannel.exists(region, key);
152         return "exists:" + exists;
153     }
154
155     /**
156      * 清理指定区域的缓存
157      *
158      * @return {@link String}
159      */
160     @GetMapping("/clear")
161     public String clear()
162     {
163         cache = null;
164         cacheChannel.clear(region);
165         return "clear success";
166     }
167
168
169     @Autowired
170     private RedisUtils redisUtils;
171
172     private Student queryMysqlById(long id)
173     {
174         log.info("查询Mysql数据库");
175         try
176         {
177             Thread.sleep(10);
178         }
179         catch (InterruptedException e)
180         {
181             e.printStackTrace();
182         }
183         Student student = new Student();
184         student.setId(id);
185         student.setName("张三");
186         return student;
187     }
188
189     /**
190      * 查询mysql ,测试缓存穿透
191      *
192      * @param id id
193      * @return {@link Student}

```

```

194     */
195     private Student queryMysqlById2(long id)
196     {
197         log.info("查询Mysql数据库2");
198         try
199         {
200             Thread.sleep(10);
201         }
202         catch (InterruptedException e)
203         {
204             e.printStackTrace();
205         }
206         return null;
207     }
208
209     private Boolean updateMysqlById(Student student, long id)
210     {
211         log.info("更新Mysql数据库");
212         try
213         {
214             Thread.sleep(10);
215         }
216         catch (InterruptedException e)
217         {
218             e.printStackTrace();
219         }
220         return true;
221     }
222
223
224
225
226     @GetMapping("/query")
227     public Student query()
228     {
229         Student result = redisUtils.query("tools:", "tools:lock:", 1L,
Student.class,
230             this::queryMysqlById, 30L, TimeUnit.MINUTES, 60);
231         return result;
232     }
233
234     @GetMapping("/query2")
235     public Student query2()
236     {
237         Student result = redisUtils.query("tools:", "tools:lock:", 2L,
Student.class,
238             this::queryMysqlById2, 30L, TimeUnit.MINUTES, 60);
239         return result;
240     }
241
242     @GetMapping("/update")
243     public boolean update()
244     {
245         Student student = new Student();
246         student.setId(1L);
247         student.setName("张三2");
248         boolean update = redisUtils.update(1L, student, "tools:", s ->
updateMysqlById(student, 1L));

```

```
249         return update;
250     }
251
252
253
254 }
255
```

第五步：启动程序

```

1      .   _       _
2    /\ / __ \ ___( )___ _ _ \\ \\ \
3  ( ( )\__ \| '_ | ' _ \| \_ \| \ \ \
4  \| \|_|_)| |_| ||||| (| |) ))))
5     '|_||. |_||_|_| |_\ , | / / / /
6  =====|_|=====|_/=//_/_/_/
7  :: Spring Boot ::                (v2.7.1)
8
9  2022-11-06 13:49:33.554 INFO 6148 --- [          main]
mao.use_starter.UseStarterApplication : Starting UseStarterApplication
using Java 1.8.0_332 on mao with PID 6148 (H:\程序\大四上期
\j2cache_spring_boot_starter_demo\use-starter\target\classes started by mao
in H:\程序\大四上期\j2cache_spring_boot_starter_demo)
10 2022-11-06 13:49:33.556 DEBUG 6148 --- [          main]
mao.use_starter.UseStarterApplication : Running with Spring Boot v2.7.1,
Spring v5.3.21
11 2022-11-06 13:49:33.556 INFO 6148 --- [          main]
mao.use_starter.UseStarterApplication : No active profile set, falling
back to 1 default profile: "default"
12 2022-11-06 13:49:33.891 INFO 6148 --- [          main]
o.s.c.a.ConfigurationClassParser      : Properties location
[${j2cache.config-location}] not resolvable: Could not resolve placeholder
'j2cache.config-location' in value "${j2cache.config-location}"
13 2022-11-06 13:49:34.361 INFO 6148 --- [          main]
.s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules
found, entering strict repository configuration mode
14 2022-11-06 13:49:34.363 INFO 6148 --- [          main]
.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data Redis
repositories in DEFAULT mode.
15 2022-11-06 13:49:34.382 INFO 6148 --- [          main]
.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository
scanning in 5 ms. Found 0 Redis repository interfaces.
16 2022-11-06 13:49:34.601 INFO 6148 --- [          main]
mao.tools_j2cache.config.CacheConfig  : 初始化 CacheConfig
17 2022-11-06 13:49:34.601 INFO 6148 --- [          main]
trationDelegate$BeanPostProcessorChecker : Bean
'mao.tools_j2cache.config.CacheConfig' of type
[mao.tools_j2cache.config.CacheConfig] is not eligible for getting processed
by all BeanPostProcessors (for example: not eligible for auto-proxying)
```

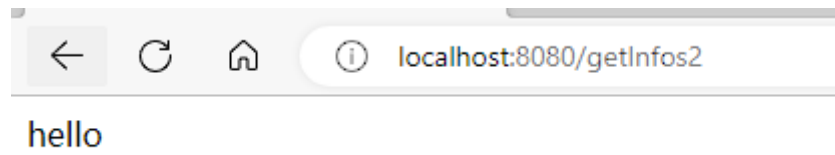
```

18 2022-11-06 13:49:34.845 INFO 6148 --- [          main]
   o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s):
   8080 (http)
19 2022-11-06 13:49:34.853 INFO 6148 --- [          main]
   o.apache.catalina.core.StandardService : Starting service [Tomcat]
20 2022-11-06 13:49:34.853 INFO 6148 --- [          main]
   org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache
   Tomcat/9.0.64]
21 2022-11-06 13:49:34.976 INFO 6148 --- [          main] o.a.c.c.C.[Tomcat].
   [localhost].[/] : Initializing Spring embedded WebApplicationContext
22 2022-11-06 13:49:34.976 INFO 6148 --- [          main]
   w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext:
   initialization completed in 1384 ms
23 2022-11-06 13:49:35.182 INFO 6148 --- [          main]
   n.o.j2cache.util.SerializationUtils : Using Serializer ->
   [json:net.oschina.j2cache.util.FstJSONSerializer]
24 2022-11-06 13:49:35.185 INFO 6148 --- [          main]
   net.oschina.j2cache.CacheProviderHolder : Using L1 CacheProvider :
   net.oschina.j2cache.caffeine.CaffeineProvider
25 2022-11-06 13:49:35.820 INFO 6148 --- [          main]
   net.oschina.j2cache.CacheProviderHolder : Using L2 CacheProvider :
   net.oschina.j2cache.cache.support.redis.SpringRedisProvider
26 2022-11-06 13:49:35.832 INFO 6148 --- [          main]
   net.oschina.j2cache.J2CacheBuilder : Using cluster policy :
   net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
27 2022-11-06 13:49:35.854 INFO 6148 --- [          main]
   mao.tools_j2cache.config.RedissonConfig : 初始化 RedissonConfig
28 2022-11-06 13:49:35.858 INFO 6148 --- [          main]
   mao.tools_j2cache.config.RedissonConfig : 单机模式redis:127.0.0.1:6379
29 2022-11-06 13:49:35.971 INFO 6148 --- [          main]
   org.redisson.Version : Redisson 3.17.0
30 2022-11-06 13:49:36.284 INFO 6148 --- [sson-netty-4-13]
   o.r.c.pool.MasterPubSubConnectionPool : 1 connections initialized for
   127.0.0.1/127.0.0.1:6379
31 2022-11-06 13:49:36.292 INFO 6148 --- [sson-netty-4-19]
   o.r.c.pool.MasterConnectionPool : 24 connections initialized for
   127.0.0.1/127.0.0.1:6379
32 2022-11-06 13:49:36.628 INFO 6148 --- [          main]
   m.tools_j2cache.config.RedisUtilsConfig : 初始化 RedisUtilsConfig
33 2022-11-06 13:49:36.885 INFO 6148 --- [          main]
   o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath
   base path '/actuator'
34 2022-11-06 13:49:36.925 INFO 6148 --- [          main]
   o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080
   (http) with context path ''
35 2022-11-06 13:49:37.206 INFO 6148 --- [          main]
   mao.use_starter.UseStarterApplication : Started UseStarterApplication in
   4.039 seconds (JVM running for 4.968)
36 2022-11-06 13:49:37.976 INFO 6148 --- [2)-172.18.144.1] o.a.c.c.C.[Tomcat].
   [localhost].[/] : Initializing Spring DispatcherServlet
   'dispatcherServlet'
37 2022-11-06 13:49:37.976 INFO 6148 --- [2)-172.18.144.1]
   o.s.web.servlet.DispatcherServlet : Initializing Servlet
   'dispatcherServlet'
38 2022-11-06 13:49:37.977 INFO 6148 --- [2)-172.18.144.1]
   o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms

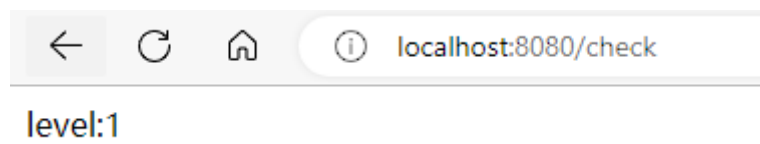
```

第六步：访问

<http://localhost:8080/getInfos2>

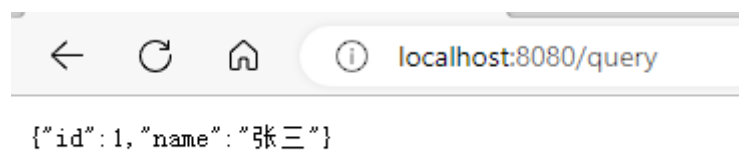


<http://localhost:8080/check>



测试并发

<http://localhost:8080/query>



String tools:1 ✕		
字符串	生存时间	对象
服务器	redis 数据库	
键	tools:1	
值	{ "id":1, "name": "张三" }	

注释:

在取样器错误后要执行的动作

☒ 继续
 ☐ 启动下一进程循环
 ☐ 停止线程
 ☐ 停止测试
 ☐ 立即停止测试

线程属性

线程数:

Ramp-Up时间(秒):

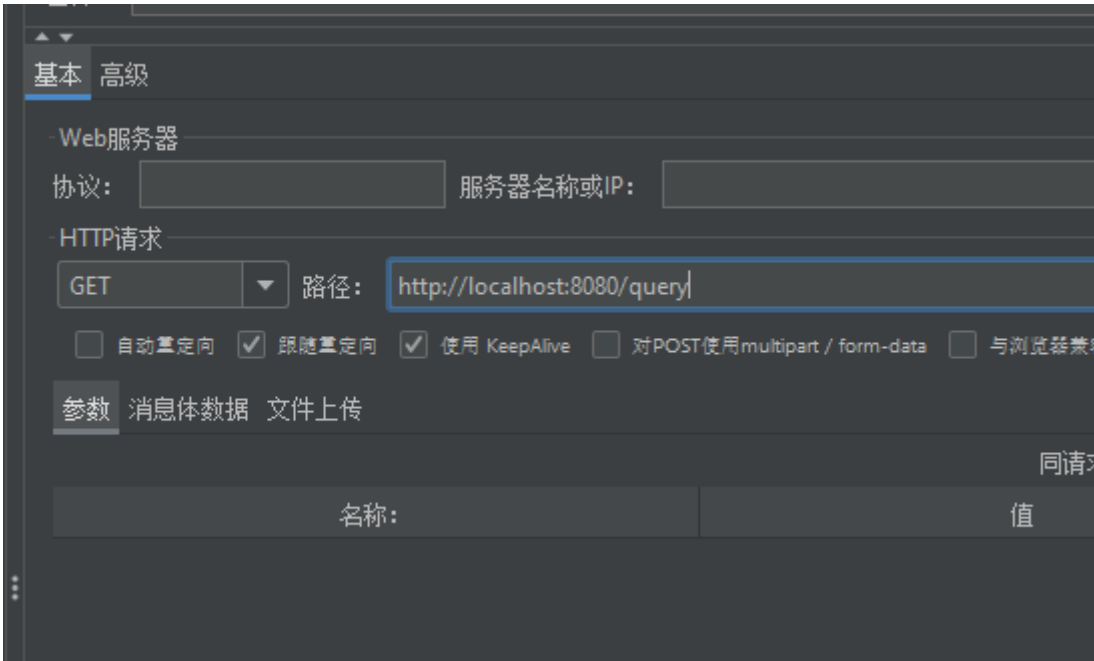
循环次数 ☒ 永远

☒ Same user on each iteration

☐ 调度器

持续时间(秒)

启动延迟(秒)



文件名

浏览...

显示日志内容:

☐ 仅错误日志

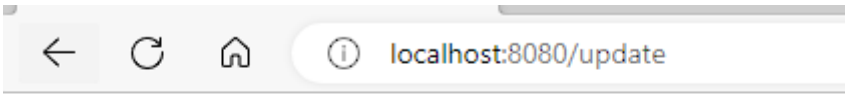
☐ 仅成功日志

配置

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	494149	9	0	594	7.53	0.00%	21228.5/sec	4078.14	2508.46	196.7
总体	494165	9	0	594	7.53	0.00%	21228.0/sec	4078.03	2508.39	196.7

尝试一次更新，让缓存过期

<http://localhost:8080/update>



true

```
[o-8080-exec-124] m.use_starter.controller.TestController : 更新MySQL数据库
[o-8080-exec-124] mao.tools_j2cache.utils.RedisUtils : 更新: tools:1
[io-8080-exec-26] mao.tools_j2cache.utils.RedisUtils : tools:1 获取分布式锁后，缓存未命中，查询数据库
[io-8080-exec-26] m.use_starter.controller.TestController : 查询MySQL数据库
```

```

1 2022-11-06 13:57:23.033 INFO 6148 --- [o-8080-exec-124]
   m.use_starter.controller.TestController : 更新Mysql数据库
2 2022-11-06 13:57:23.062 DEBUG 6148 --- [o-8080-exec-124]
   mao.tools_j2cache.utils.RedisUtils : 更新: tools:1
3 2022-11-06 13:57:23.197 DEBUG 6148 --- [io-8080-exec-26]
   mao.tools_j2cache.utils.RedisUtils : tools:1 获取分布式锁后, 缓存未命中, 查询
   数据库
4 2022-11-06 13:57:23.197 INFO 6148 --- [io-8080-exec-26]
   m.use_starter.controller.TestController : 查询Mysql数据库

```

缓存过期后, 只更新了一次数据库

测试缓存穿透

<http://localhost:8080/query2>

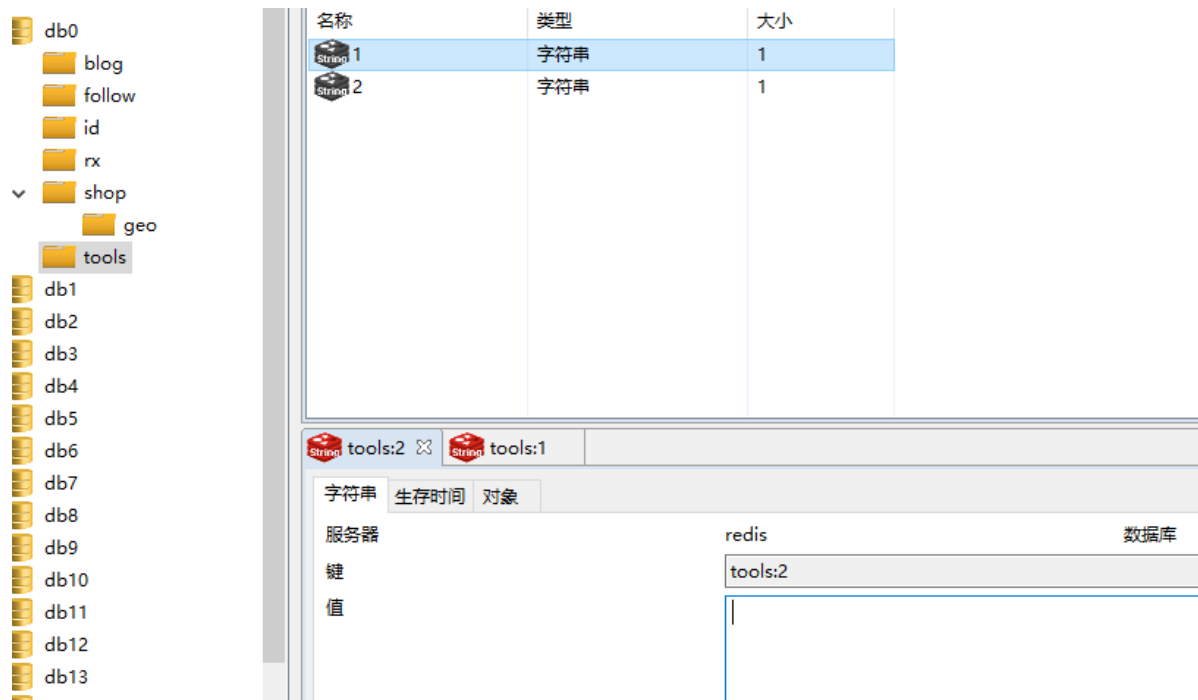
多次访问

```

2022-11-06 14:03:35.136 INFO 4628 --- [1]-172.18.144.1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-11-06 14:03:35.139 INFO 4628 --- [1]-172.18.144.1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
2022-11-06 14:03:51.216 DEBUG 4628 --- [nio-8080-exec-2] mao.tools_j2cache.utils.RedisUtils : tools:2 获取分布式锁后, 缓存未命中, 查询数据库
2022-11-06 14:03:51.216 INFO 4628 --- [nio-8080-exec-2] m.use_starter.controller.TestController : 查询Mysql数据库2

```

数据库只查询了一次



缓存的是一个空字符串

```
1 C:\Users\mao>redis-cli
2 127.0.0.1:6379> auth 123456
3 OK
4 127.0.0.1:6379> get tools:2
5 ""
6 127.0.0.1:6379>
```

end

by mao
2022 11 06
