第六步：访问

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

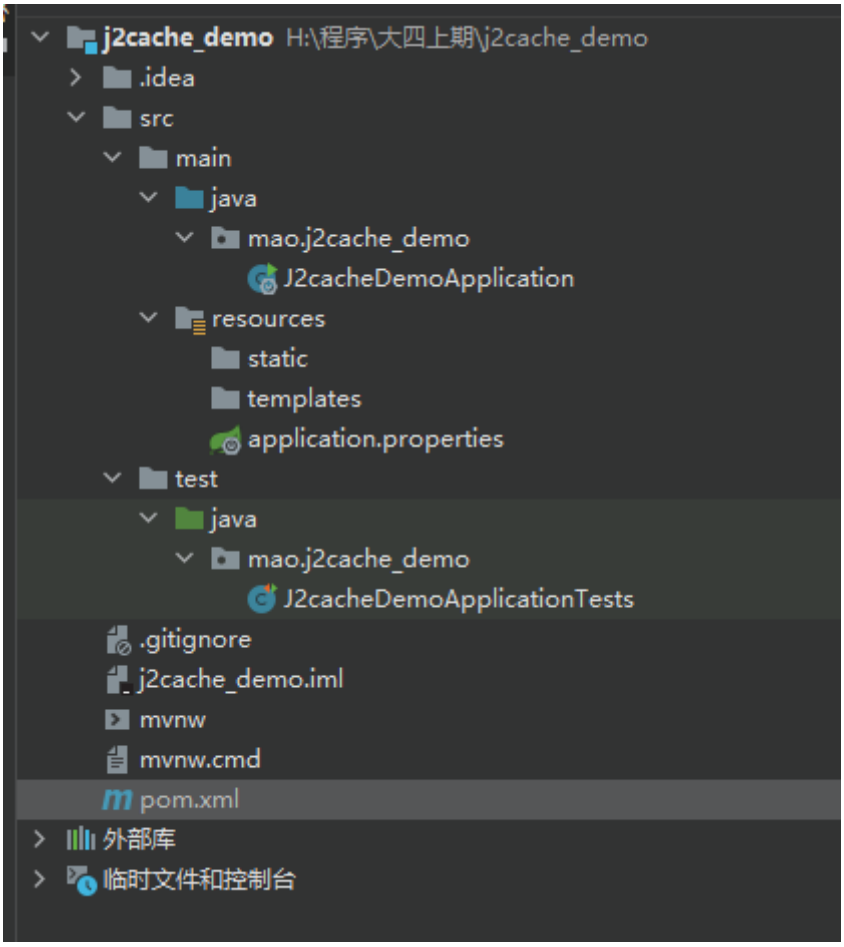
使用j2cache需要导入的maven坐标：

```
<dependency>
    <groupId>net.oschina.j2cache</groupId>
    <artifactId>j2cache-spring-boot2-starter</artifactId>
    <version>2.8.0-release</version>
</dependency>
<dependency>
    <groupId>net.oschina.j2cache</groupId>
```

```
    <artifactId>j2cache-core</artifactId>
    <version>2.8.0-release</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

j2cache入门案例

第一步：创建工程j2cache_demo



第二步：修改pom文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>mao</groupId>
    <artifactId>j2cache_demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>j2cache_demo</name>
    <description>j2cache_demo</description>
    <properties>
        <java.version>8</java.version>
    </properties>
    <dependencies>

        <dependency>
```

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>net.oschina.j2cache</groupId>
            <artifactId>j2cache-spring-boot2-starter</artifactId>
            <version>2.8.0-release</version>
        </dependency>
        <dependency>
            <groupId>net.oschina.j2cache</groupId>
            <artifactId>j2cache-core</artifactId>
            <version>2.8.0-release</version>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-simple</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-api</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第三步：修改application.yml文件

```
spring:
  cache:
    type: GENERIC
  redis:
```

```
    host: 127.0.0.1
    password: 123456
    port: 6379
    database: 0

j2cache:
  #  config-location: /j2cache.properties
  open-spring-cache: true
  cache-clean-mode: passive
  allow-null-values: true
  redis-client: lettuce #指定redis客户端使用lettuce，也可以使用Jedis
  l2-cache-open: true #开启二级缓存
  broadcast: net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
  #  broadcast: jgroups
  L1: #指定一级缓存提供者为caffeine
    provider_class: caffeine
  L2: #指定二级缓存提供者为redis
    provider_class:
net.oschina.j2cache.cache.support.redis.SpringRedisProvider
    config_section: lettuce
  sync_ttl_to_redis: true
  default_cache_null_object: false
  serialization: fst
caffeine:
  properties: /caffeine.properties    # 这个配置文件需要放在项目中
lettuce:
  mode: single
  namespace:
  storage: generic
  channel: j2cache
  scheme: redis
  hosts: 127.0.0.1:6379
  password: 123456
  database: 0
  sentinelMasterId:
  maxTotal: 100
  maxIdle: 10
  minIdle: 10
  timeout: 10000
```

第四步：启动Redis

```
C:\Users\mao>redis-cli
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

第五步：创建/resources/caffeine.properties文件

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

第六步：编写TestController

```
package mao.j2cache_demo.controller;

import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.CacheObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;

/**
 * Project name(项目名称)：j2cache_demo
 * Package(包名): mao.j2cache_demo.controller
 * Class(类名): TestController
 * Author(作者）: mao
 * Author QQ: 1296193245
 * GitHub: https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 13:22
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
public class TestController
{

    @Autowired
    private CacheChannel cacheChannel;

    private final String key = "myKey";
    private final String region = "rx";

```

```
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}
```

```
     */
    @GetMapping("/clear")
    public String clear()
    {
        cacheChannel.clear(region);
        return "clear success";
    }
}
```

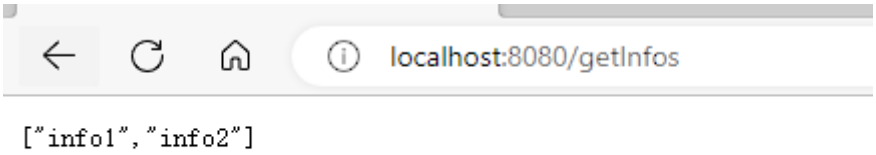
第七步：启动程序

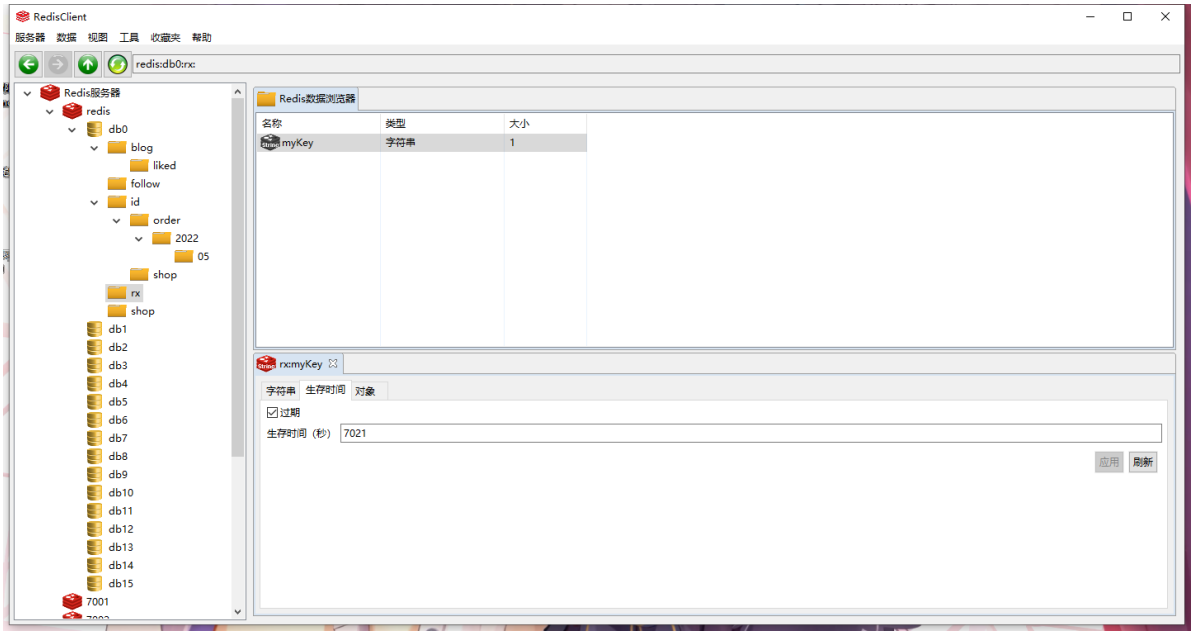
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

2022-11-05 13:43:46.628  INFO 10704 --- [           main] mao.j2cache_demo.J2cacheDemoApplication  : Starting J2cacheDemoApplication using Java 1.8.0_332 on mao with PID 10704 (H:\程序\大四上期\j2cache_demo\target\classes started by mao in H:\程序\大四上期\j2cache_demo)
2022-11-05 13:43:46.630  INFO 10704 --- [           main] mao.j2cache_demo.J2cacheDemoApplication  : No active profile set, falling back to 1 default profile: "default"
2022-11-05 13:43:46.895  INFO 10704 --- [           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}"
2022-11-05 13:43:47.087  INFO 10704 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules found, entering strict repository configuration mode
2022-11-05 13:43:47.089  INFO 10704 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data Redis repositories in DEFAULT mode.
2022-11-05 13:43:47.106  INFO 10704 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 5 ms. Found 0 Redis repository interfaces.
2022-11-05 13:43:47.462  INFO 10704 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-11-05 13:43:47.469  INFO 10704 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-11-05 13:43:47.469  INFO 10704 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-11-05 13:43:47.580  INFO 10704 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
```

```
2022-11-05 13:43:47.580  INFO 10704 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 918 ms
2022-11-05 13:43:47.671  INFO 10704 --- [           main] n.o.j2cache.util.SerializationUtils      : Using Serializer -> [fst:net.oschina.j2cache.util.FSTSerializer]
2022-11-05 13:43:47.674  INFO 10704 --- [           main] net.oschina.j2cache.CacheProviderHolder  : Using L1 CacheProvider : net.oschina.j2cache.caffeine.CaffeineProvider
2022-11-05 13:43:47.864  INFO 10704 --- [           main] net.oschina.j2cache.CacheProviderHolder  : Using L2 CacheProvider : net.oschina.j2cache.cache.support.redis.SpringRedisProvider
2022-11-05 13:43:47.873  INFO 10704 --- [           main] net.oschina.j2cache.J2CacheBuilder       : Using cluster policy : net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
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 ''
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)
```

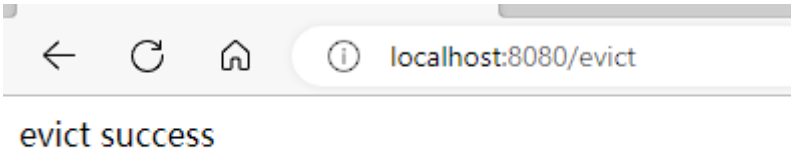
第八步：访问

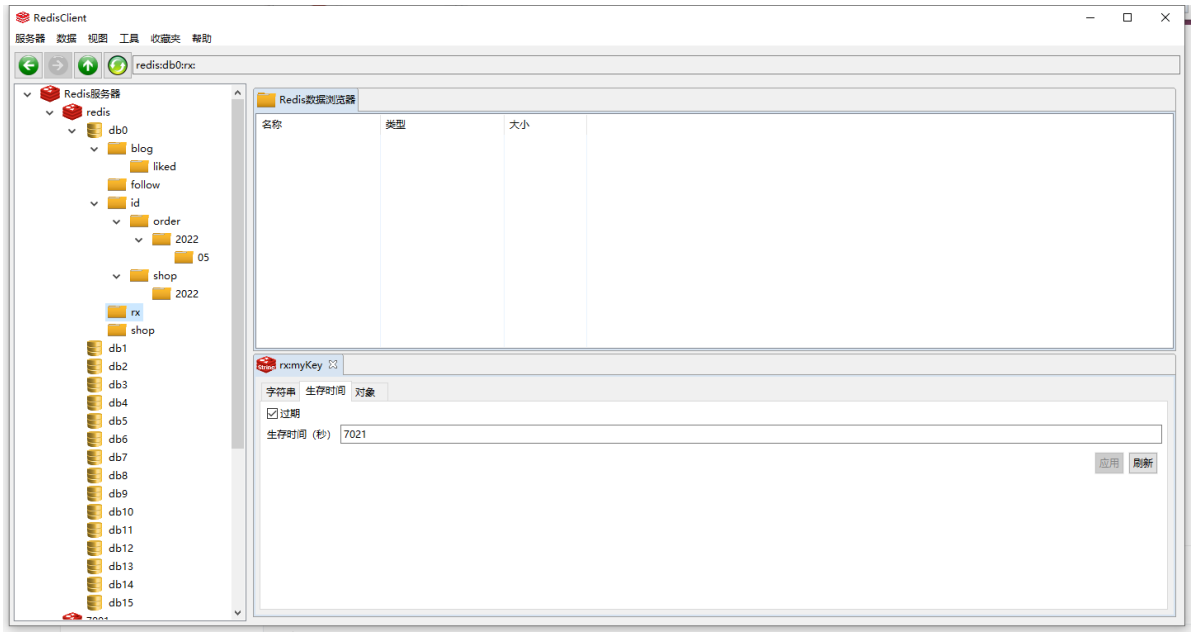
http://localhost:8080/getInfos





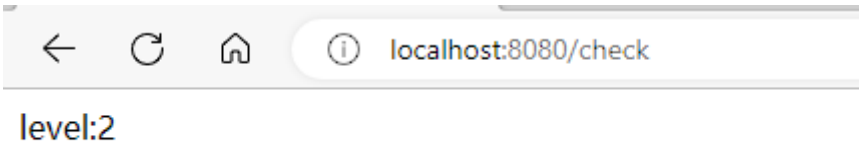
http://localhost:8080/evict



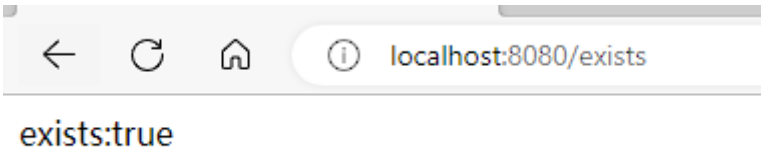


http://localhost:8080/getInfos

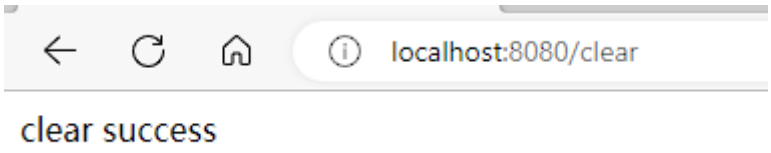
http://localhost:8080/check

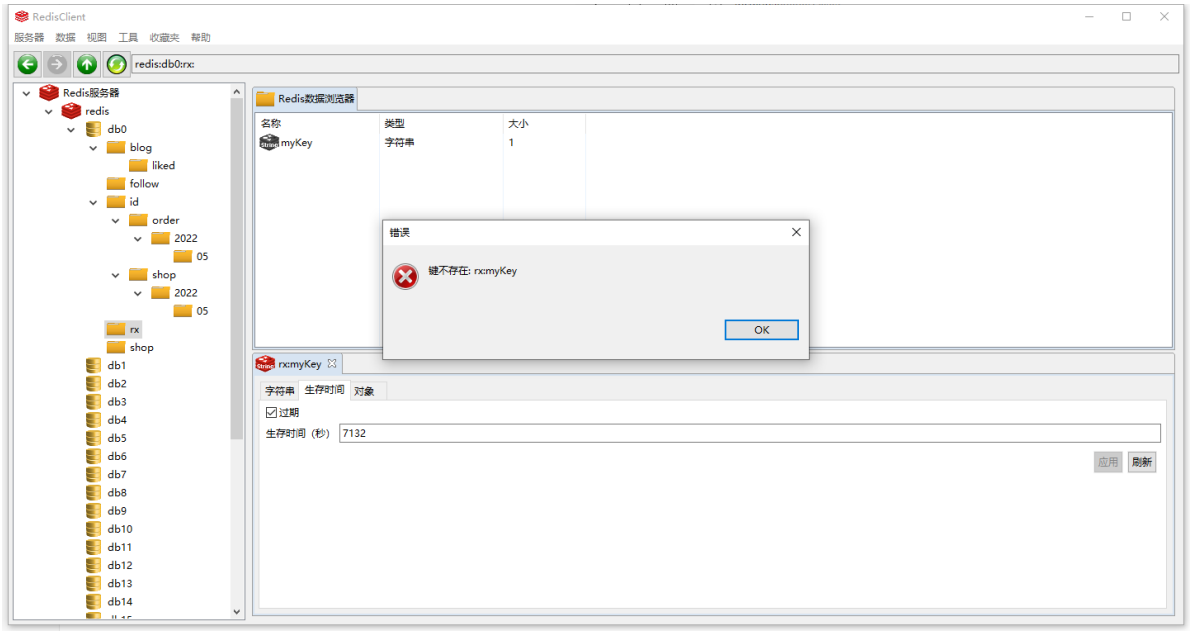


http://localhost:8080/exists



http://localhost:8080/clear





无法启动解决

启动报以下错误:

```
Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.
2022-11-05 13:32:38.028 ERROR 8520 --- [           main] o.s.boot.SpringApplication               : Application run failed

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
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.resolveFieldValue(AutowiredAnnotationBeanPostProcessor.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(Abstr
actApplicationContext.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]
```

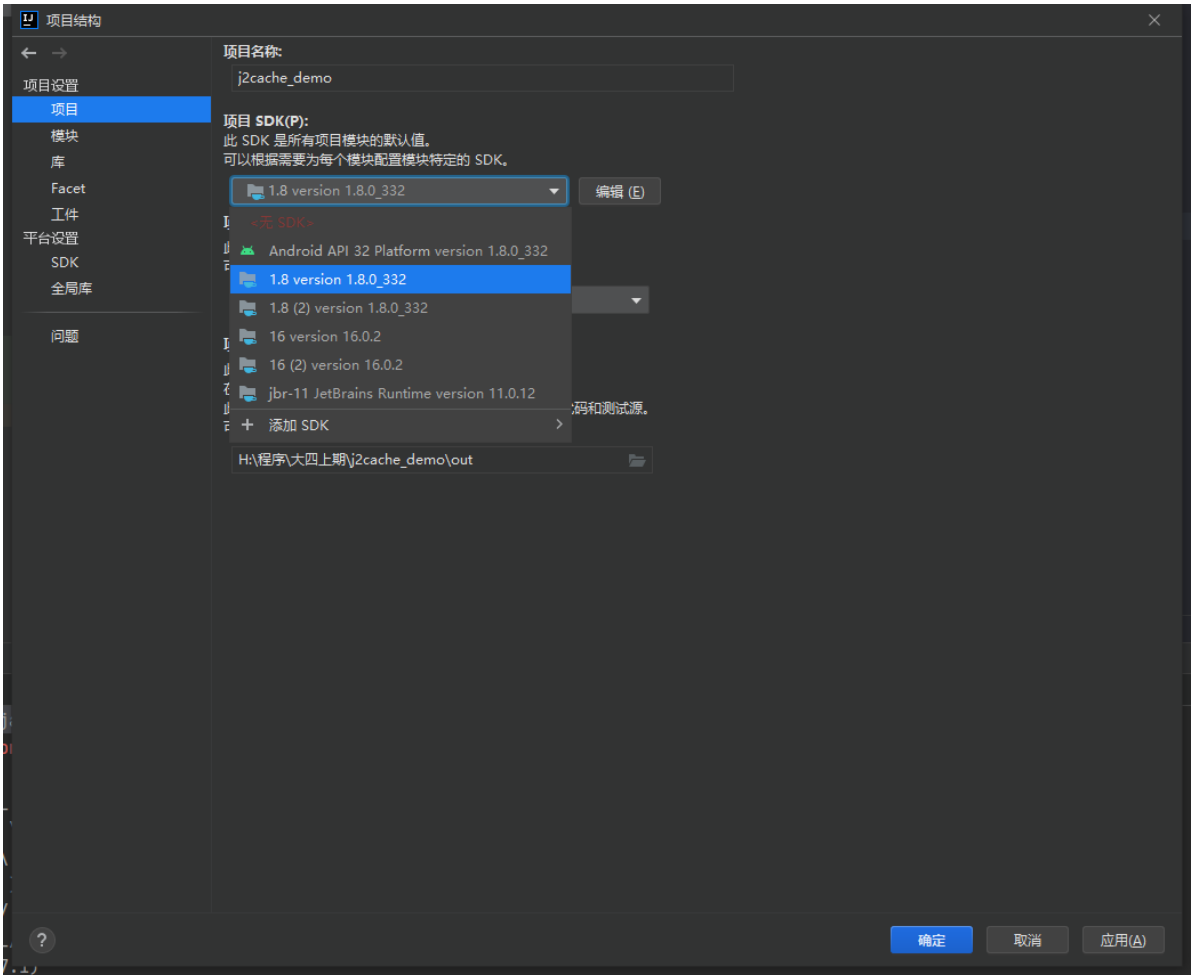
```
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:308) ~[spring-boot-2.7.1.jar:2.7.1]
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1306) ~[spring-boot-2.7.1.jar:2.7.1]
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1295) ~[spring-boot-2.7.1.jar:2.7.1]
    at mao.j2cache_demo.J2cacheDemoApplication.main(J2cacheDemoApplication.java:12) ~[classes/:na]
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
    at org.springframework.beans.factory.support.ConstructorResolver.instantiate(ConstructorResolver.java:658) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.ConstructorResolver.instantiateUsingFactoryMethod(ConstructorResolver.java:638) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.instantiateUsingFactoryMethod(AbstractAutowireCapableBeanFactory.java:1352) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBeanInstance(AbstractAutowireCapableBeanFactory.java:1195) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:582) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:542) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:335) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:234) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:333) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:208) ~[spring-beans-5.3.21.jar:5.3.21]
```

```
    at org.springframework.beans.factory.config.DependencyDescriptor.resolveCandidate(DependencyDescriptor.java:276) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:1391) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:1311) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.resolveFieldValue(AutowiredAnnotationBeanPostProcessor.java:656) ~[spring-beans-5.3.21.jar:5.3.21]
    ... 20 common frames omitted
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
    at org.springframework.beans.factory.support.SimpleInstantiationStrategy.instantiate(SimpleInstantiationStrategy.java:185) ~[spring-beans-5.3.21.jar:5.3.21]
    at org.springframework.beans.factory.support.ConstructorResolver.instantiate(ConstructorResolver.java:653) ~[spring-beans-5.3.21.jar:5.3.21]
    ... 33 common frames omitted
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
    at java.base/java.lang.reflect.AccessibleObject.checkCanSetAccessible(AccessibleObject.java:357) ~[na:na]
    at java.base/java.lang.reflect.AccessibleObject.checkCanSetAccessible(AccessibleObject.java:297) ~[na:na]
    at java.base/java.lang.reflect.Field.checkCanSetAccessible(Field.java:177) ~[na:na]
    at java.base/java.lang.reflect.Field.setAccessible(Field.java:171) ~[na:na]
    at org.nustaq.serialization.FSTClazzInfo.createFieldInfo(FSTClazzInfo.java:512) ~[fst-2.57.jar:na]
    at org.nustaq.serialization.FSTClazzInfo.createFields(FSTClazzInfo.java:368) ~[fst-2.57.jar:na]
    at org.nustaq.serialization.FSTClazzInfo.<init>(FSTClazzInfo.java:129) ~[fst-2.57.jar:na]
    at org.nustaq.serialization.FSTClazzInfoRegistry.getCLInfo(FSTClazzInfoRegistry.java:129) ~[fst-2.57.jar:na]
```

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

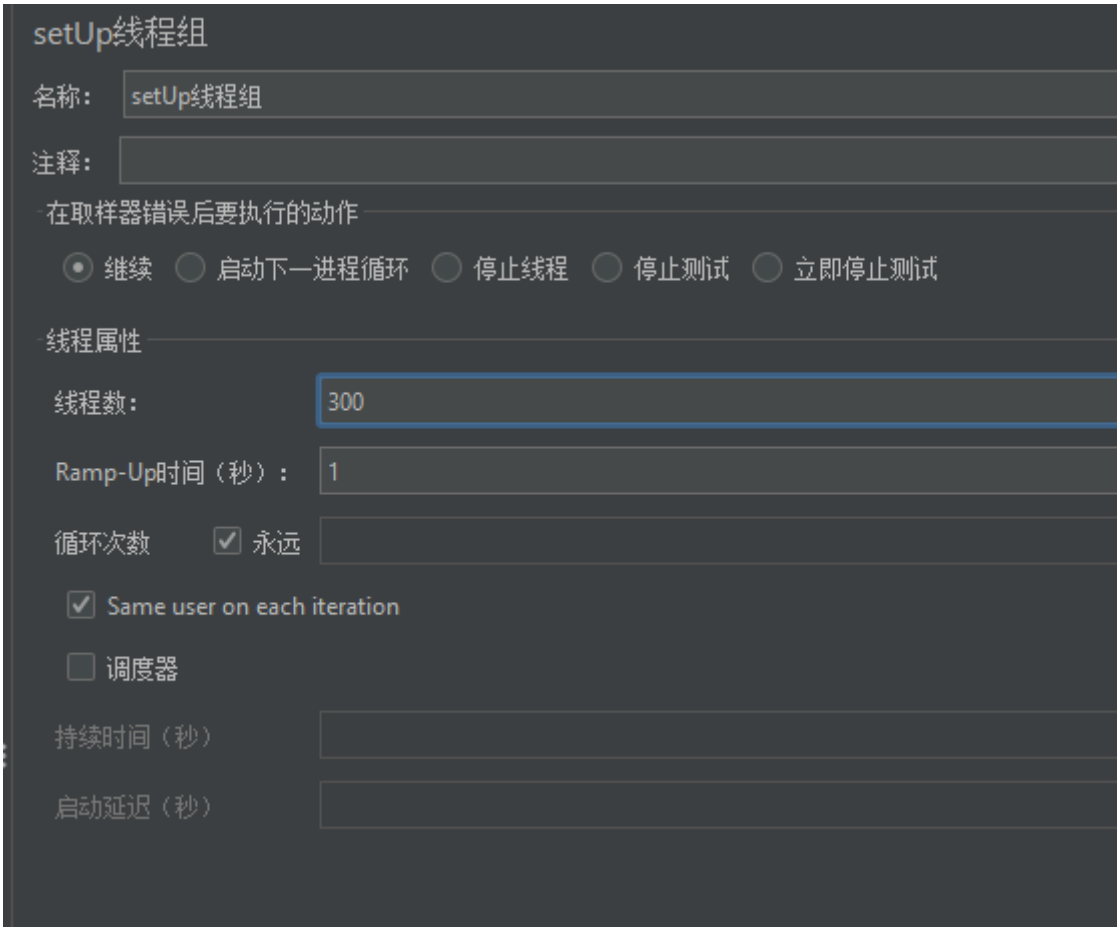
第一步：修改TestController

```java
package mao.j2cache_demo.controller;

import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.CacheObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;

/**
 * Project name(项目名称)：j2cache_demo
 * Package(包名): mao.j2cache_demo.controller
 * Class(类名): TestController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 13:22
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
public class TestController
{

    private static final Logger log =
LoggerFactory.getLogger(TestController.class);

    @Autowired
    private CacheChannel cacheChannel;

    private final String key = "myKey";
    private final String region = "rx";


    @GetMapping("/getInfos")
    public List<String> getInfos()
    {
        CacheObject cacheObject = cacheChannel.get(region, key);
        if (cacheObject.getValue() == null)
```

```
        {
            log.info("查询数据库");
            //缓存中没有找到，查询数据库获得
            List<String> data = new ArrayList<>();
            data.add("info1");
            data.add("info2");
            try
            {
                Thread.sleep(9);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            //放入缓存
            cacheChannel.set(region, key, data);
            return data;
        }
        return (List<String>) cacheObject.getValue();
    }

    /**
     * 清理指定缓存
     *
     * @return {@link String}
     */
    @GetMapping("/evict")
    public String evict()
    {
        cacheChannel.evict(region, key);
        return "evict success";
    }

    /**
     * 检测存在哪级缓存
     *
     * @return {@link String}
     */
    @GetMapping("/check")
    public String check()
    {
        int check = cacheChannel.check(region, key);
        return "level:" + check;
    }

    /**
     * 检测缓存数据是否存在
     *
     * @return {@link String}
     */
    @GetMapping("/exists")
    public String exists()
    {
        boolean exists = cacheChannel.exists(region, key);
        return "exists:" + exists;
    }

    /**
```

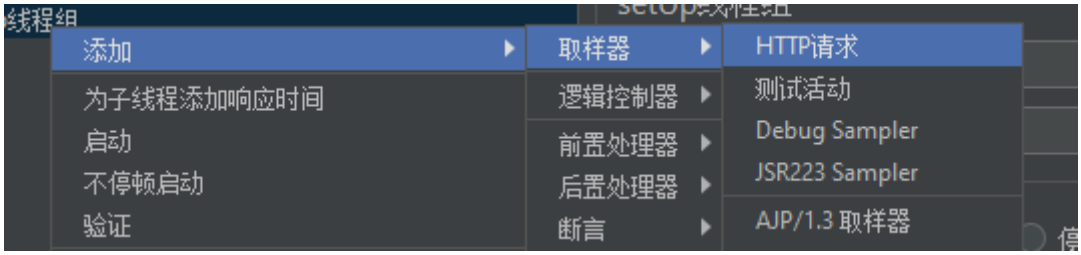
```
103         * 清理指定区域的缓存
104         *
105         * @return {@link String}
106         */
107        @GetMapping("/clear")
108        public String clear()
109        {
110            cacheChannel.clear(region);
111            return "clear success";
112        }
113    }
```

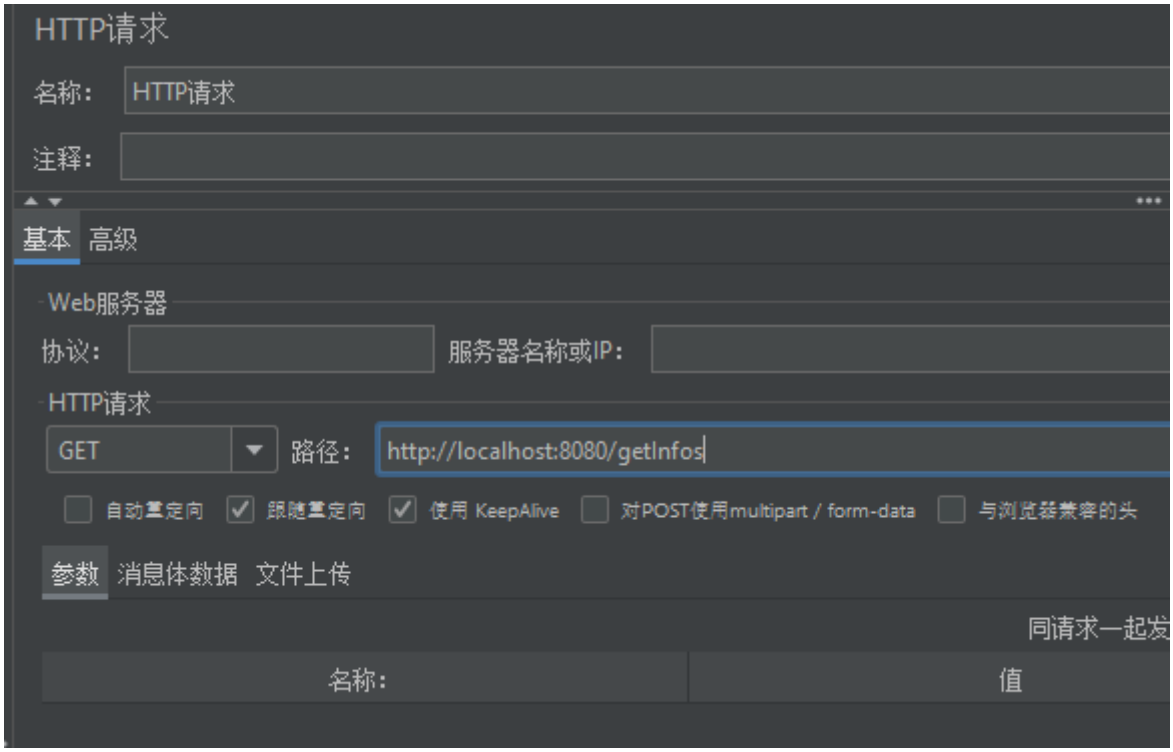
第二步：启动程序，并打开jmeter



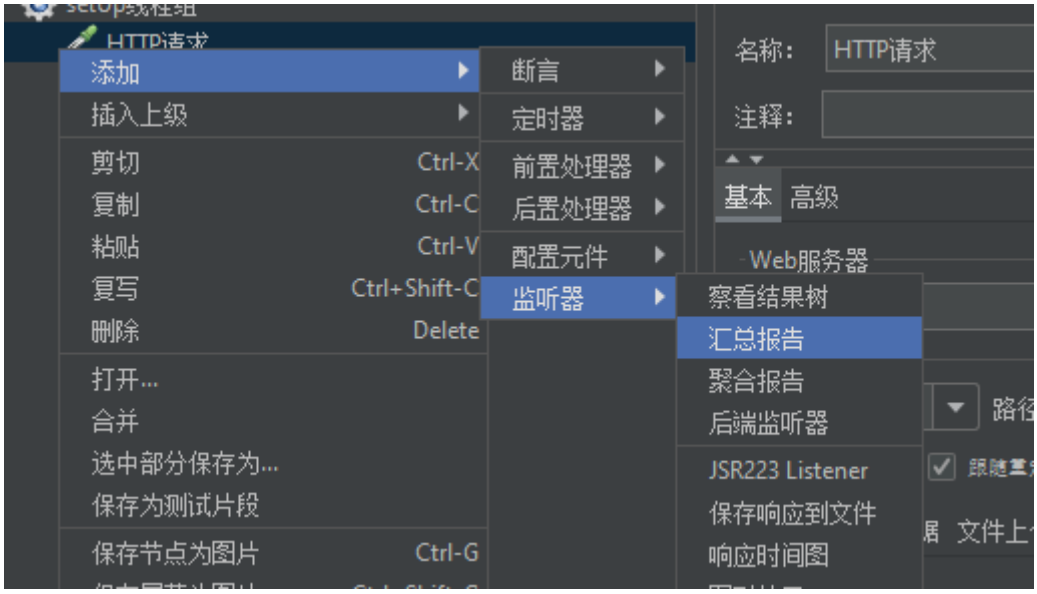
300线程并发

第三步：设置http请求

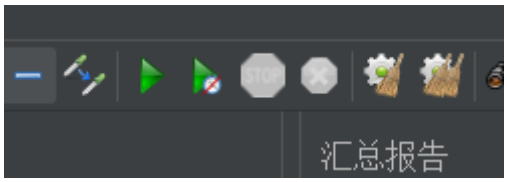




第四步：添加监听器

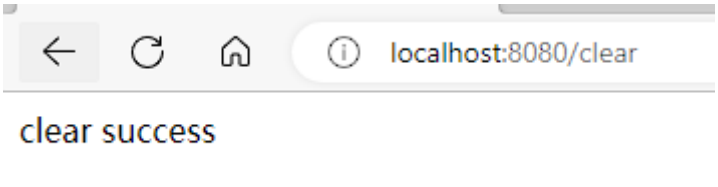


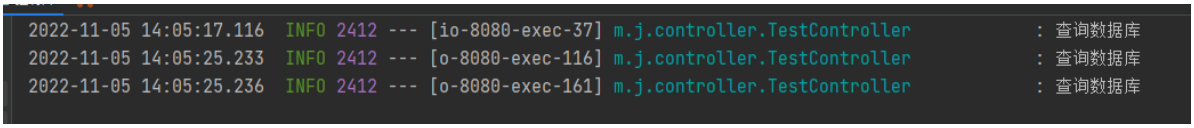
第五步：启动jmeter



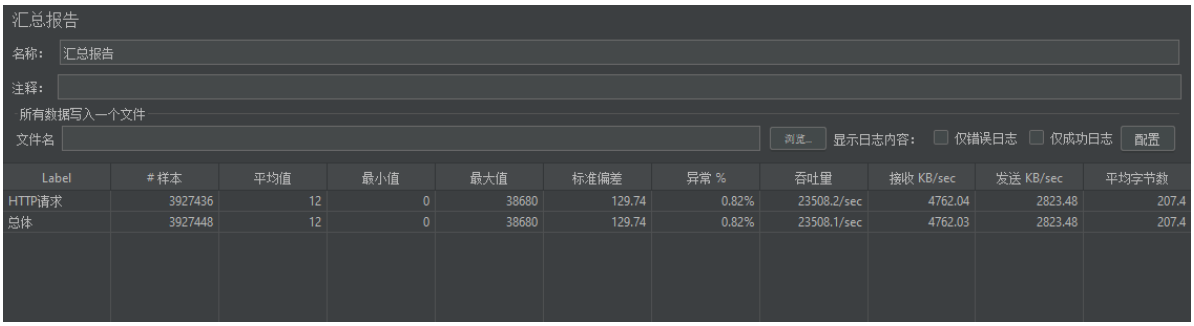
第六步：主动清除缓存

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
```



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

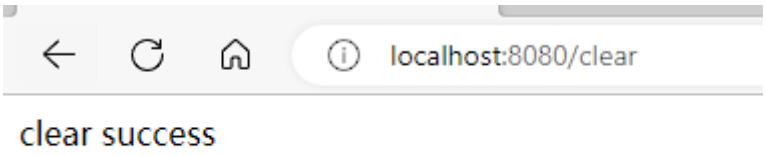
有时候能同时通过两个请求，有时候能同时通过3个请求，没有完全解决缓存击穿问题，但是影响不大

拿自己实现的缓存做比较

```java
package mao.j2cache_demo.controller;

import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.CacheObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;

/**
 * Project name(项目名称)：j2cache_demo
 * Package(包名): mao.j2cache_demo.controller
 * Class(类名): TestController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 13:22
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
public class TestController
```

```
{

    private static final Logger log =
LoggerFactory.getLogger(TestController.class);

    @Autowired
    private CacheChannel cacheChannel;

    private final String key = "myKey";
    private final String region = "rx";


    @GetMapping("/getInfos")
    public List<String> getInfos()
    {
        CacheObject cacheObject = cacheChannel.get(region, key);
        if (cacheObject.getValue() == null)
        {
            log.info("查询数据库");
            //缓存中没有找到，查询数据库获得
            List<String> data = new ArrayList<>();
            data.add("info1");
            data.add("info2");
            //放入缓存
            cacheChannel.set(region, key, data);
            return data;
        }
        return (List<String>) cacheObject.getValue();
    }

    private String cache = null;

    @GetMapping("/getInfos2")
    public String getInfos2()
    {
        if (cache == null)
        {
            log.info("查询数据库2");
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            cache = "hello";
        }
        else
        {
            return cache;
        }
        return cache;
    }

    /**
     * 清理指定缓存
     *
```

```
     * @return {@link String}
     */
    @GetMapping("/evict")
    public String evict()
    {
        cacheChannel.evict(region, key);
        return "evict success";
    }

    /**
     * 检测存在哪级缓存
     *
     * @return {@link String}
     */
    @GetMapping("/check")
    public String check()
    {
        int check = cacheChannel.check(region, key);
        return "level:" + check;
    }

    /**
     * 检测缓存数据是否存在
     *
     * @return {@link String}
     */
    @GetMapping("/exists")
    public String exists()
    {
        boolean exists = cacheChannel.exists(region, key);
        return "exists:" + exists;
    }

    /**
     * 清理指定区域的缓存
     *
     * @return {@link String}
     */
    @GetMapping("/clear")
    public String clear()
    {
        cache = null;
        cacheChannel.clear(region);
        return "clear success";
    }
}
```




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

```
2022-11-05 14:20:07.152  INFO 4668 --- [nio-8080-exec-6] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [io-8080-exec-10] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [io-8080-exec-71] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [io-8080-exec-94] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [io-8080-exec-99] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [o-8080-exec-195] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [o-8080-exec-192] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [io-8080-exec-49] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [io-8080-exec-76] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [o-8080-exec-170] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.152  INFO 4668 --- [o-8080-exec-140] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-13] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-190] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-26] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-31] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-159] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-123] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-90] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-61] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-179] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-200] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-35] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-60] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-135] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-83] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-63] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-37] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-84] m.j.controller.TestController           : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-23] m.j.controller.TestController           : 查询数据库2
```

```
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-85] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-184] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-97] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-93] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-89] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-51] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-38] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-194] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-177] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-197] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-33] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-120] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-160] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-108] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-17] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-96] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-12] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-22] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-98] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-72] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [nio-8080-exec-4] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [io-8080-exec-28] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-171] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-91] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-151] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-34] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-110] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-154] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-174] m.j.controller.TestController            : 查询数据库2
```

```
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-64] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-186] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-161] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-66] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [io-8080-exec-73] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-54] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-193] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [io-8080-exec-70] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-39] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-57] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [io-8080-exec-42] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-79] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-56] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-47] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-67] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-44] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-80] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-191] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-122] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-130] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-86] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-166] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-136] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-169] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-125] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-137] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-129] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [nio-8080-exec-7] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [nio-8080-exec-1] m.j.controller.TestController            : 查询数据库2
```

```
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-43] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-139] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-78] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-118] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-172] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-145] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-68] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-74] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-162] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.153  INFO 4668 --- [o-8080-exec-152] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-55] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-48] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-149] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-115] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-30] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-188] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-146] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [io-8080-exec-14] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-143] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-180] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-132] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [nio-8080-exec-8] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-58] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-142] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-92] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-165] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-87] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-173] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.154  INFO 4668 --- [o-8080-exec-114] m.j.controller.TestController            : 查询数据库2
```

```
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-164] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-156] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-126] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [nio-8080-exec-9] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-148] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-104] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-15] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-182] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [nio-8080-exec-3] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-21] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-147] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-102] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.155  INFO 4668 --- [o-8080-exec-155] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-128] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-106] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-199] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-107] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [io-8080-exec-32] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-11] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-196] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-153] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [io-8080-exec-82] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-36] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-178] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [io-8080-exec-16] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-18] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-109] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [io-8080-exec-65] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.156  INFO 4668 --- [o-8080-exec-141] m.j.controller.TestController            : 查询数据库2
```

```
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-189] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-46] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-52] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-41] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-27] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-138] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-144] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-20] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-117] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [o-8080-exec-134] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.157  INFO 4668 --- [io-8080-exec-45] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [nio-8080-exec-5] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-113] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-62] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-133] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-183] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-88] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-101] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [nio-8080-exec-2] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-185] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-121] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-95] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-81] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-175] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-163] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-112] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-187] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [o-8080-exec-168] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.158  INFO 4668 --- [io-8080-exec-25] m.j.controller.TestController            : 查询数据库2
```

```
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-124] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-119] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-75] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-29] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-59] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-105] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-50] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-19] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-158] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-127] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-167] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [io-8080-exec-77] m.j.controller.TestController            : 查询数据库2
2022-11-05 14:20:07.159  INFO 4668 --- [o-8080-exec-181] m.j.controller.TestController            : 查询数据库2
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

```java
package mao.j2cache_demo.controller;

import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.CacheObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
```

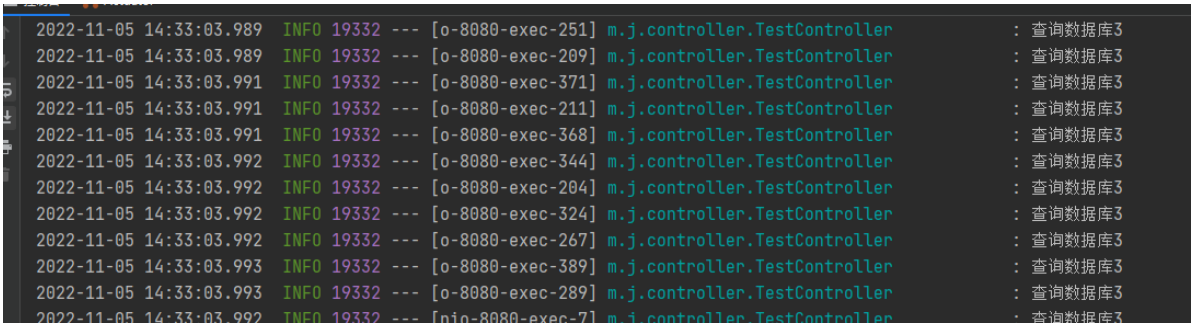
```java
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;

/**
 * Project name(项目名称)：j2cache_demo
 * Package(包名): mao.j2cache_demo.controller
 * Class(类名): TestController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 13:22
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
public class TestController
{

    private static final Logger log =
LoggerFactory.getLogger(TestController.class);

    @Autowired
    private CacheChannel cacheChannel;

    private final String key = "myKey";
    private final String region = "rx";


    @GetMapping("/getInfos")
    public List<String> getInfos()
    {
        CacheObject cacheObject = cacheChannel.get(region, key);
        if (cacheObject.getValue() == null)
        {
            log.info("查询数据库");
            //缓存中没有找到，查询数据库获得
            List<String> data = new ArrayList<>();
            data.add("info1");
            data.add("info2");
            try
            {
                Thread.sleep(9);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            //放入缓存
            cacheChannel.set(region, key, data);
            return data;
        }
        return (List<String>) cacheObject.getValue();
    }

```

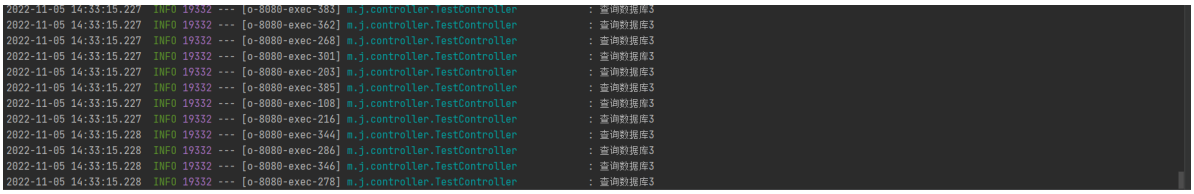
```
66      private String cache = null;
67
68      @GetMapping("/getInfos2")
69      public String getInfos2()
70      {
71          if (cache == null)
72          {
73              log.info("查询数据库2");
74              try
75              {
76                  Thread.sleep(10);
77              }
78              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);
```

```
        return "evict success";
    }

    /**
     * 检测存在哪级缓存
     *
     * @return {@link String}
     */
    @GetMapping("/check")
    public String check()
    {
        int check = cacheChannel.check(region, key);
        return "level:" + check;
    }

    /**
     * 检测缓存数据是否存在
     *
     * @return {@link String}
     */
    @GetMapping("/exists")
    public String exists()
    {
        boolean exists = cacheChannel.exists(region, key);
        return "exists:" + exists;
    }

    /**
     * 清理指定区域的缓存
     *
     * @return {@link String}
     */
    @GetMapping("/clear")
    public String clear()
    {
        cache = null;
        cacheChannel.clear(region);
        return "clear success";
    }
}
```

第二步：设置http请求

第三步：重启服务并启动jmeter





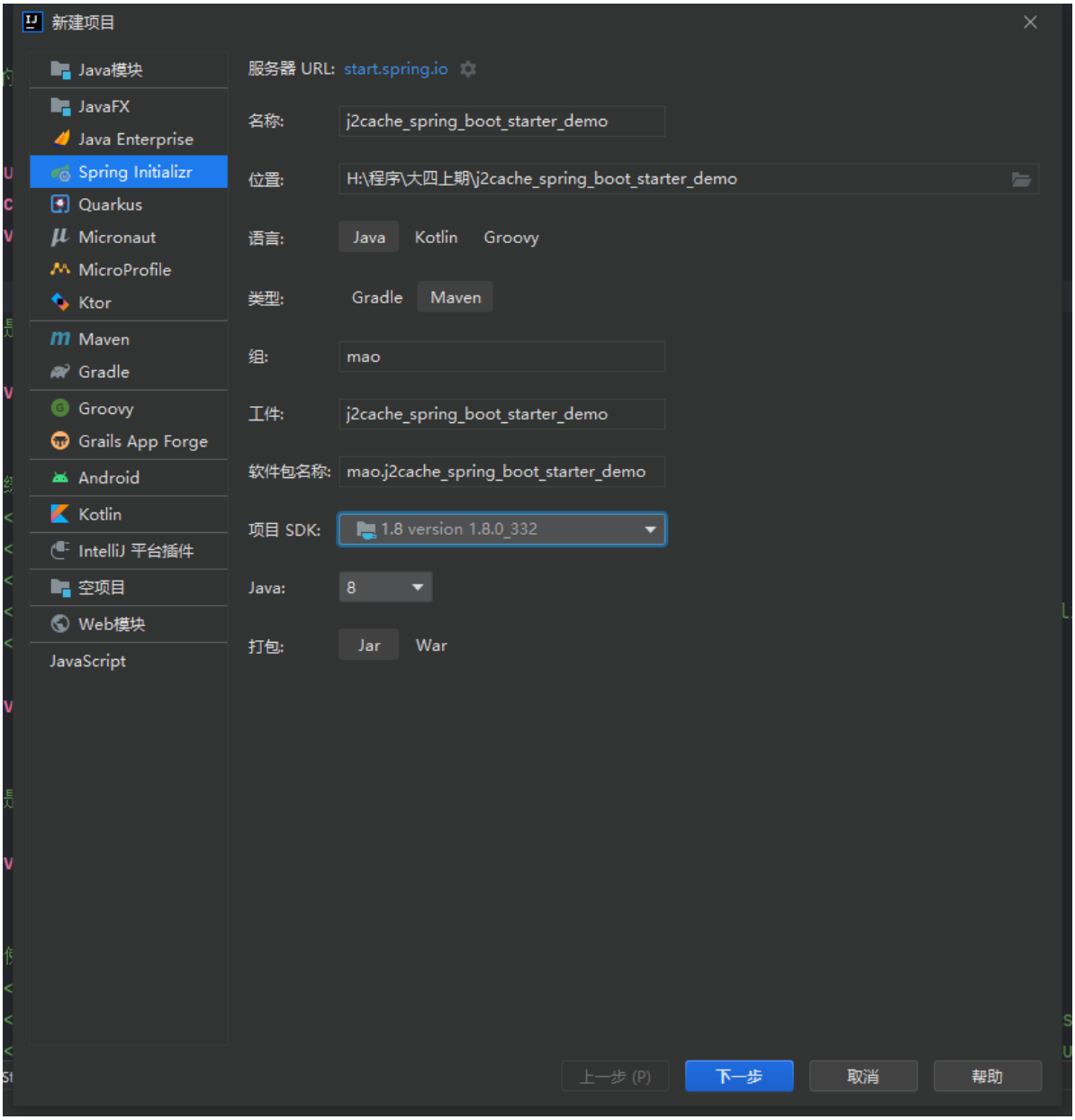
所以，使用缓存要小心，缓存击穿、缓存雪崩和缓存穿透需要自己解决

自定义spring boot starter

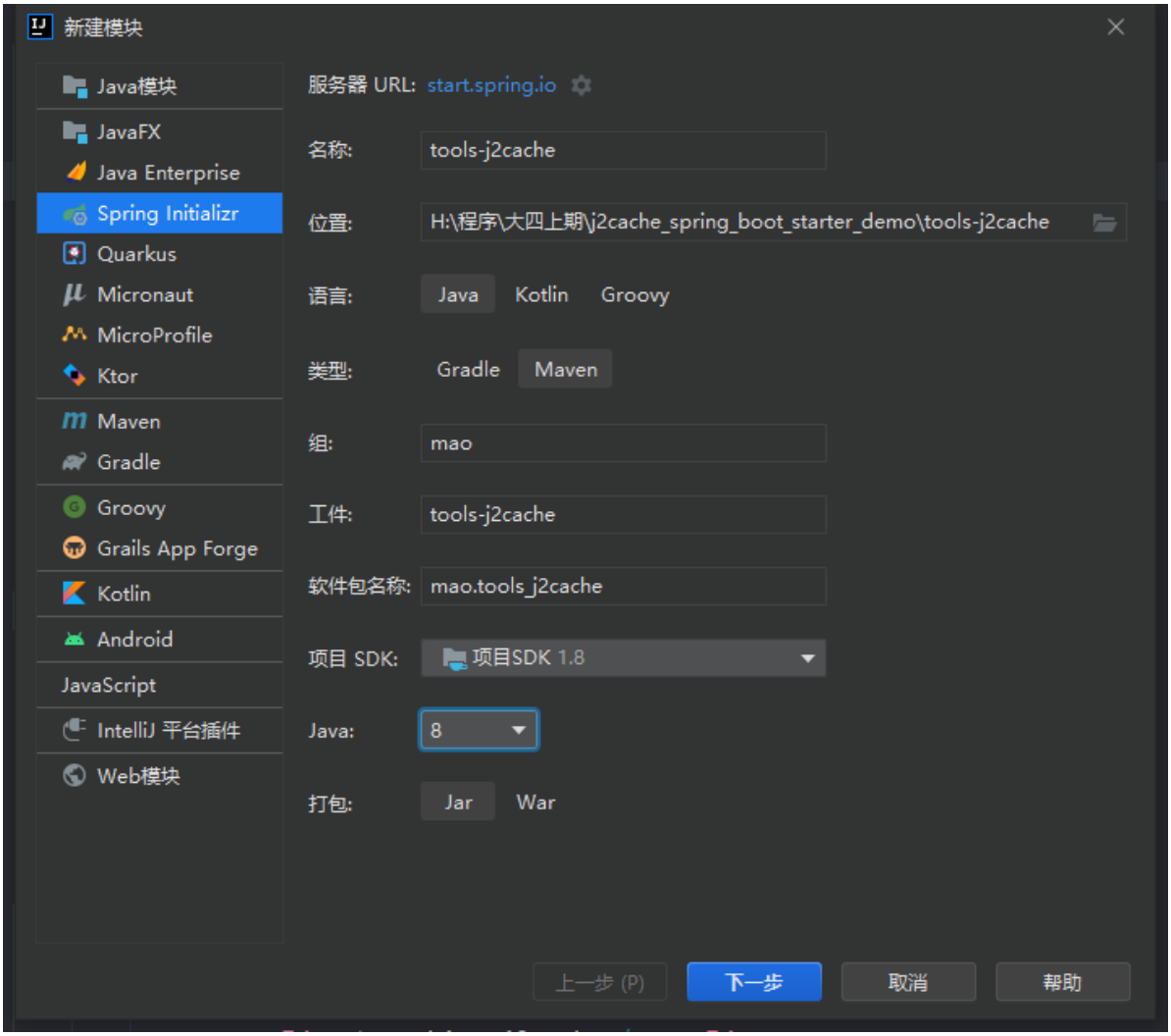
开发starter

第一步：初始化项目

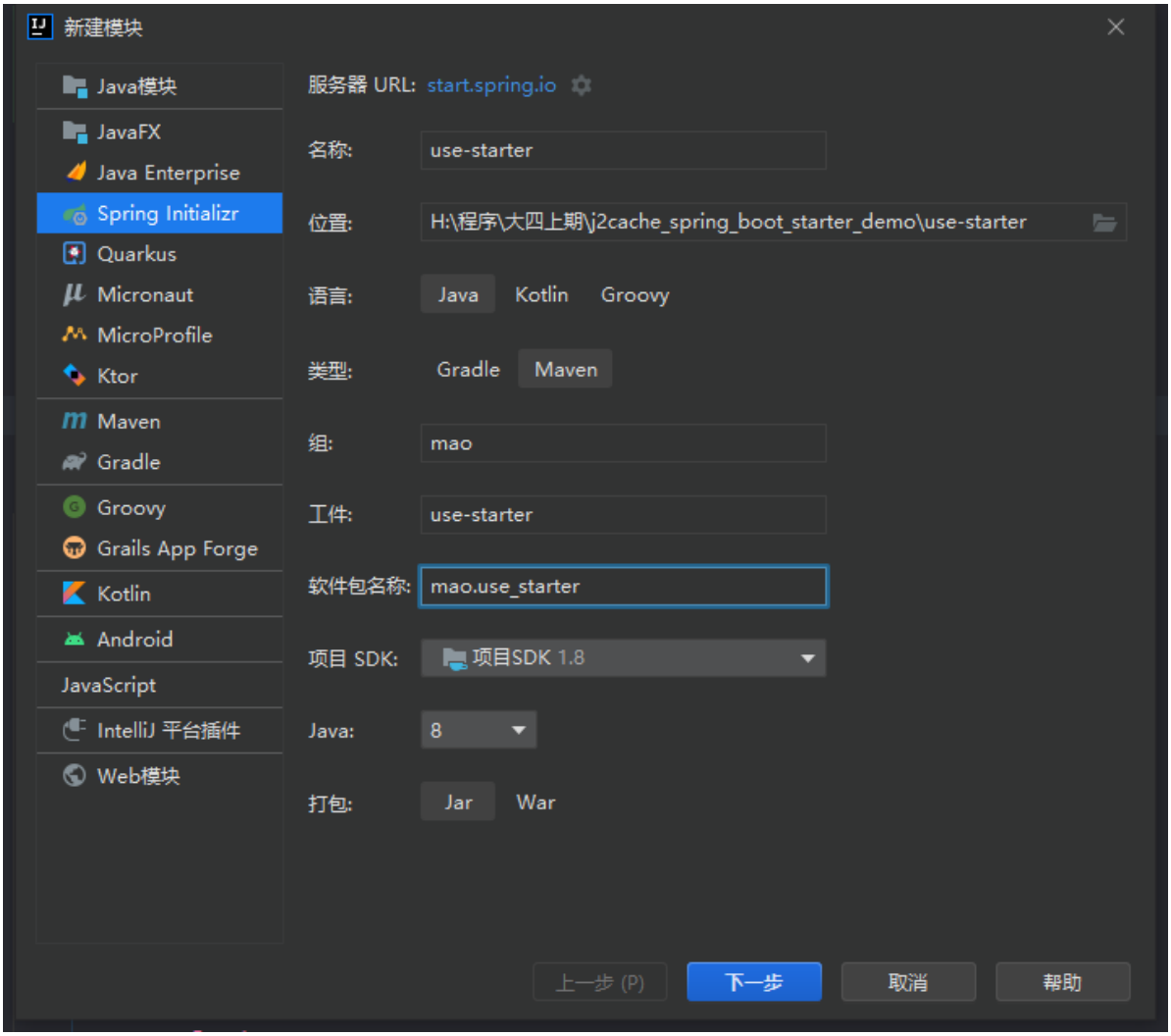
创建父工程j2cache_spring_boot_starter_demo



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：

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

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>mao</groupId>
```

```
    <artifactId>j2cache_spring_boot_starter_demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>j2cache_spring_boot_starter_demo</name>
    <description>j2cache_spring_boot_starter_demo</description>
    <packaging>pom</packaging>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>

    </dependencies>

    <modules>
        <module>tools-j2cache</module>
        <module>use-starter</module>
    </modules>

    <dependencyManagement>
        <dependencies>

            <dependency>
                <groupId>net.oschina.j2cache</groupId>
                <artifactId>j2cache-spring-boot2-starter</artifactId>
                <version>2.8.0-release</version>
            </dependency>

            <dependency>
                <groupId>net.oschina.j2cache</groupId>
                <artifactId>j2cache-core</artifactId>
                <version>2.8.0-release</version>
            </dependency>

        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>

```

子工程tools-j2cache的pom文件：

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

```
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>j2cache_spring_boot_starter_demo</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <artifactId>tools-j2cache</artifactId>
    <name>tools-j2cache</name>
    <description>tools-j2cache</description>

    <properties>

    </properties>

    <dependencies>

        <!--spring boot starter开发依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-autoconfigure</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
        </dependency>

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

        <dependency>
            <groupId>net.oschina.j2cache</groupId>
            <artifactId>j2cache-spring-boot2-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>net.oschina.j2cache</groupId>
            <artifactId>j2cache-core</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-simple</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.objenesis</groupId>
                    <artifactId>objenesis</artifactId>
                </exclusion>
                <exclusion>
                    <artifactId>javassist</artifactId>
```

```
                    <groupId>org.javassist</groupId>
                </exclusion>
                <exclusion>
                    <artifactId>fastjson</artifactId>
                    <groupId>com.alibaba</groupId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.jgroups</groupId>
            <artifactId>jgroups</artifactId>
            <version>3.6.15.Final</version>
        </dependency>

        <dependency>
            <artifactId>javassist</artifactId>
            <groupId>org.javassist</groupId>
            <version>3.25.0-GA</version>
        </dependency>

        <dependency>
            <groupId>org.objenesis</groupId>
            <artifactId>objenesis</artifactId>
            <version>2.6</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
        </dependency>
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.9.2</version>
        </dependency>
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.2</version>
```

```
        </dependency>
        <dependency>
            <groupId>aopalliance</groupId>
            <artifactId>aopalliance</artifactId>
            <version>1.0</version>
        </dependency>

        <!--阿里巴巴的FastJson json解析-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.79</version>
        </dependency>

        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>5.8.0</version>
        </dependency>

        <!--spring boot redisson 依赖-->
        <dependency>
            <groupId>org.redisson</groupId>
            <artifactId>redisson-spring-boot-starter</artifactId>
            <version>3.17.0</version>
        </dependency>

    </dependencies>

</project>

```

子工程use-starter的pom文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>j2cache_spring_boot_starter_demo</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <artifactId>use-starter</artifactId>
    <name>use-starter</name>
    <description>use-starter</description>

    <properties>

    </properties>
```

```
    <dependencies>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第三步：修改类J2CacheCache

```
package net.oschina.j2cache.cache.support;

import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.CacheObject;
import net.oschina.j2cache.NullObject;
import org.springframework.cache.CacheManager;
import org.springframework.cache.support.AbstractValueAdaptingCache;
import org.springframework.cache.support.NullValue;

import java.util.concurrent.Callable;

/**
 * {@link CacheManager} implementation for J2Cache.
 */
public class J2CacheCache extends AbstractValueAdaptingCache
{

    private CacheChannel cacheChannel;

    private String j2CacheName = "j2cache";

```

```
22      public J2CacheCache(String cacheName, CacheChannel cacheChannel)
23      {
24          this(cacheName, cacheChannel, true);
25      }
26
27      public J2CacheCache(String cacheName, CacheChannel cacheChannel,
    boolean allowNullValues)
28      {
29          super(allowNullValues);
30          j2CacheName = cacheName;
31          this.cacheChannel = cacheChannel;
32      }
33
34      @Override
35      public String getName()
36      {
37          return this.j2CacheName;
38      }
39
40      public void setJ2CacheNmae(String name)
41      {
42          this.j2CacheName = name;
43      }
44
45      @Override
46      public Object getNativeCache()
47      {
48          return this.cacheChannel;
49      }
50
51      @Override
52      public <T> T get(Object key, Callable<T> valueLoader)
53      {
54          T value;
55          try
56          {
57              value = valueLoader.call();
58          }
59          catch (Exception ex)
60          {
61              throw new ValueRetrievalException(key, valueLoader, ex);
62          }
63          put(key, value);
64          return value;
65      }
66
67      @Override
68      public void put(Object key, Object value)
69      {
70          cacheChannel.set(j2CacheName, String.valueOf(key), value,
    super.isAllowNullValues());
71      }
72
73      @Override
74      public ValueWrapper putIfAbsent(Object key, Object value)
75      {
76          if (!cacheChannel.exists(j2CacheName, String.valueOf(key)))
77          {
```

```
            cacheChannel.set(j2CacheName, String.valueOf(key), value);
        }
        return get(key);
    }

    @Override
    public void evict(Object key)
    {
        cacheChannel.evict(j2CacheName, String.valueOf(key));
    }

    @Override
    public void clear()
    {
        cacheChannel.clear(j2CacheName);
    }

    @Override
    protected Object lookup(Object key)
    {
        CacheObject cacheObject = cacheChannel.get(j2CacheName,
String.valueOf(key));
        if (cacheObject.rawValue() != null &&
cacheObject.rawValue().getClass().equals(NullObject.class) &&
super.isAllowNullValues())
        {
            return NullValue.INSTANCE;
        }
        return cacheObject.getValue();
    }

}
```

第四步：修改类J2CacheCacheManger

```
package net.oschina.j2cache.cache.support;

import java.util.Collection;
import java.util.Collections;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;

import net.oschina.j2cache.CacheChannel;
import org.springframework.cache.Cache;
import
org.springframework.cache.transaction.AbstractTransactionSupportingCacheMana
ger;
import org.springframework.util.CollectionUtils;

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
```

```
     */
    public void setAllowNullValues(boolean allowNullValues)
    {
        this.allowNullValues = allowNullValues;
    }

    @Override
    protected Cache getMissingCache(String name)
    {
        return this.dynamic ? new J2CacheCache(name, cacheChannel,
allowNullValues) : null;
    }


    /**
     * 设置缓存名称
     *
     * @param cacheNames 缓存名称
     */
    public void setCacheNames(Collection<String> cacheNames)
    {
        Set<String> newCacheNames = CollectionUtils.isEmpty(cacheNames) ?
Collections.emptySet()
                : new HashSet<>(cacheNames);
        this.cacheNames = newCacheNames;
        this.dynamic = newCacheNames.isEmpty();
    }

}
```

第五步：添加类J2CacheSerializer

```
package net.oschina.j2cache.cache.support.util;

import net.oschina.j2cache.util.SerializationUtils;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.data.redis.serializer.SerializationException;

import java.io.IOException;


public class J2CacheSerializer implements RedisSerializer<Object>
{

    @Override
    public byte[] serialize(Object t) throws SerializationException
    {
        try
        {
            return SerializationUtils.serialize(t);
        }
```

```
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public Object deserialize(byte[] bytes) throws SerializationException
    {
        try
        {
            return SerializationUtils.deserialize(bytes);
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

}
```

第六步：添加类SpringJ2CacheConfigUtil

```
package net.oschina.j2cache.cache.support.util;

import net.oschina.j2cache.J2CacheConfig;
import org.springframework.core.env.CompositePropertySource;
import org.springframework.core.env.EnumerablePropertySource;
import org.springframework.core.env.MapPropertySource;
import org.springframework.core.env.StandardEnvironment;

public class SpringJ2CacheConfigUtil
{

    /**
     * 从spring环境变量中查找j2cache配置
     */
    public static J2CacheConfig initFromConfig(StandardEnvironment
environment)
    {
        J2CacheConfig config = new J2CacheConfig();

 config.setSerialization(environment.getProperty("j2cache.serialization"));
        config.setBroadcast(environment.getProperty("j2cache.broadcast"));

 config.setL1CacheName(environment.getProperty("j2cache.L1.provider_class")
);
```

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

```
                String[] propertyNames = new String[]{};
                if (ps instanceof CompositePropertySource)
                {
                    CompositePropertySource cps = (CompositePropertySource)
ps;
                    propertyNames = cps.getPropertyNames();
                }
                else if (ps instanceof EnumerablePropertySource)
                {
                    EnumerablePropertySource eps =
(EnumerablePropertySource) ps;
                    propertyNames = eps.getPropertyNames();
                }
                setProperty(config, environment, l2_section,
propertyNames);
            });
        }
        return config;
    }

    private static void setProperty(J2CacheConfig config,
StandardEnvironment environment, String l2_section, String[] propertyNames)
    {
        for (String key : propertyNames)
        {
            if (key.startsWith(config.getBroadcast() + "."))
            {

 config.getBroadcastProperties().setProperty(key.substring((config.getBroad
cast() + ".").length()),
                        environment.getProperty(key));
            }
            if (key.startsWith(config.getL1CacheName() + "."))
            {

 config.getL1CacheProperties().setProperty(key.substring((config.getL1Cache
Name() + ".").length()),
                        environment.getProperty(key));
            }
            if (key.startsWith(l2_section + "."))
            {

 config.getL2CacheProperties().setProperty(key.substring((l2_section +
".").length()),
                        environment.getProperty(key));
            }
        }

    }
}
```

第七步：添加类SpringUtil

```
package net.oschina.j2cache.cache.support.util;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;

/**
 * spring 工具类
 */
public class SpringUtil implements ApplicationContextAware
{

    /**
     * 应用程序上下文
     */
    private static ApplicationContext applicationContext;

    /**
     * 获取applicationContext
     */
    public static ApplicationContext getApplicationContext()
    {
        return applicationContext;
    }

    @Override
    public void setApplicationContext(ApplicationContext applicationContext)
throws BeansException
    {
        if
(net.oschina.j2cache.cache.support.util.SpringUtil.applicationContext ==
null)
        {

 net.oschina.j2cache.cache.support.util.SpringUtil.applicationContext =
applicationContext;
        }
    }

    /**
     * 通过name获取 Bean.
     */
    public static Object getBean(String name)
    {
        return getApplicationContext().getBean(name);
    }

    /**
     * 通过class获取Bean.
     */
    public static <T> T getBean(Class<T> clazz)
    {
        return getApplicationContext().getBean(clazz);
```

```
    }

    /**
     * 通过name,以及Clazz返回指定的Bean
     */
    public static <T> T getBean(String name, Class<T> clazz)
    {
        return getApplicationContext().getBean(name, clazz);
    }

}
```

第八步：添加类ConfigureNotifyKeyspaceEventsAction

```
package net.oschina.j2cache.cache.support.redis;

import org.springframework.dao.InvalidDataAccessApiUsageException;
import org.springframework.data.redis.connection.RedisConnection;

import java.util.Properties;

/**
 * 设置redis键值回调
 */
public class ConfigureNotifyKeyspaceEventsAction
{

    /**
     * 配置用于事件通知
     */
    static final String CONFIG_NOTIFY_KEYSPACE_EVENTS = "notify-keyspace-
events";


    /**
     * 配置
     *
     * @param connection 连接
     */
    public void config(RedisConnection connection)
    {
        String notifyOptions = getNotifyOptions(connection);
        String customizedNotifyOptions = notifyOptions;
        if (!customizedNotifyOptions.contains("E"))
        {
            customizedNotifyOptions += "E";
        }
        boolean A = customizedNotifyOptions.contains("A");
        if (!(A || customizedNotifyOptions.contains("g")))
        {
            customizedNotifyOptions += "g";
```

```
        }
        if (!(A || customizedNotifyOptions.contains("x")))
        {
            customizedNotifyOptions += "x";
        }
        if (!notifyOptions.equals(customizedNotifyOptions))
        {
            connection.setConfig(CONFIG_NOTIFY_KEYSPACE_EVENTS,
customizedNotifyOptions);
        }
    }

    /**
     * 得到通知选项
     *
     * @param connection 连接
     * @return {@link String}
     */
    private String getNotifyOptions(RedisConnection connection)
    {
        try
        {
            Properties config =
connection.getConfig(CONFIG_NOTIFY_KEYSPACE_EVENTS);
            if (config.isEmpty())
            {
                return "";
            }
            return
config.getProperty(config.stringPropertyNames().iterator().next());
        }
        catch (InvalidDataAccessApiUsageException e)
        {
            throw new IllegalStateException(
                    "Unable to configure Redis to keyspace notifications.
See http://docs.spring.io/spring-session/docs/current/reference/html5/#api-
redisoperationssessionrepository-sessiondestroyedevent",
                    e);
        }
    }
}
```

第九步：添加类SpringRedisActiveMessageListener

```
package net.oschina.j2cache.cache.support.redis;

import net.oschina.j2cache.cluster.ClusterPolicy;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.connection.Message;
import org.springframework.data.redis.connection.MessageListener;
```

```java

/**
 * 监听二缓key失效，主动清除本地缓存
 */
public class SpringRedisActiveMessageListener implements MessageListener
{

    /**
     * 日志记录器
     */
    private static Logger logger =
LoggerFactory.getLogger(net.oschina.j2cache.cache.support.redis.SpringRedisA
ctiveMessageListener.class);

    /**
     * 集群政策
     */
    private ClusterPolicy clusterPolicy;

    /**
     * 名称空间
     */
    private String namespace;

    SpringRedisActiveMessageListener(ClusterPolicy clusterPolicy, String
namespace)
    {
        this.clusterPolicy = clusterPolicy;
        this.namespace = namespace;
    }

    @Override
    public void onMessage(Message message, byte[] pattern)
    {
        String key = message.toString();
        if (key == null)
        {
            return;
        }
        if (key.startsWith(namespace + ":"))
        {
            String[] k = key.replaceFirst(namespace + ":", "").split(":",
2);
            if (k.length != 2)
            {
                return;
            }
            clusterPolicy.evict(k[0], k[1]);
        }

    }

}
```

第十步：添加类SpringRedisCache

```
package net.oschina.j2cache.cache.support.redis;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Set;

import net.oschina.j2cache.Level2Cache;
import org.springframework.data.redis.core.RedisCallback;
import org.springframework.data.redis.core.RedisTemplate;

/**
 * 重新实现二级缓存，采用hash结构缓存数据
 */
public class SpringRedisCache implements Level2Cache
{

    /**
     * 名称空间
     */
    private String namespace;

    /**
     * 地区
     */
    private String region;

    private RedisTemplate<String, Serializable> redisTemplate;

    public SpringRedisCache(String namespace, String region,
RedisTemplate<String, Serializable> redisTemplate)
    {
        if (region == null || region.isEmpty())
        {
            region = "_"; // 缺省region
        }
        this.namespace = namespace;
        this.redisTemplate = redisTemplate;
        this.region = getRegionName(region);
    }

    private String getRegionName(String region)
    {
        if (namespace != null && !namespace.isEmpty())
        {
            region = namespace + ":" + region;
        }
        return region;
    }

    @Override
```

```
    public void clear()
    {
        redisTemplate.opsForHash().delete(region);
    }

    @Override
    public boolean exists(String key)
    {
        return redisTemplate.opsForHash().hasKey(region, key);
    }

    @Override
    public void evict(String... keys)
    {
        for (String k : keys)
        {
            if (!k.equals("null"))
            {
                redisTemplate.opsForHash().delete(region, k);
            }
            else
            {
                redisTemplate.delete(region);
            }
        }
    }

    @Override
    public Collection<String> keys()
    {
        Set<Object> list = redisTemplate.opsForHash().keys(region);
        List<String> keys = new ArrayList<>(list.size());
        for (Object object : list)
        {
            keys.add((String) object);
        }
        return keys;
    }

    @Override
    public byte[] getBytes(String key)
    {
        return
redisTemplate.opsForHash().getOperations().execute((RedisCallback<byte[]>)
redis ->
                redis.hGet(region.getBytes(), key.getBytes()));
    }

    @Override
    public List<byte[]> getBytes(Collection<String> keys)
    {
        return
redisTemplate.opsForHash().getOperations().execute((RedisCallback<List<byte
[]>>) redis ->
        {
            byte[][] bytes = keys.stream().map(k ->
k.getBytes()).toArray(byte[][]::new);
            return redis.hMGet(region.getBytes(), bytes);
```

```
        });
    }

    @Override
    public void put(String key, Object value)
    {
        redisTemplate.opsForHash().put(region, key, value);
    }

    /**
     * 设置缓存数据的有效期
     */
    @Override
    public void put(String key, Object value, long timeToLiveInSeconds)
    {
        redisTemplate.opsForHash().put(region, key, value);
    }

    @Override
    public void setBytes(String key, byte[] bytes)
    {

 redisTemplate.opsForHash().getOperations().execute((RedisCallback<List<byte[]>>) redis ->
        {
            redis.set(_key(key).getBytes(), bytes);
            redis.hSet(region.getBytes(), key.getBytes(), bytes);
            return null;
        });
    }

    @Override
    public void setBytes(Map<String, byte[]> bytes)
    {
        bytes.forEach((k, v) ->
        {
            setBytes(k, v);
        });
    }

    private String _key(String key)
    {
        return this.region + ":" + key;
    }

}
```

第十一步：添加类SpringRedisGenericCache

```
package net.oschina.j2cache.cache.support.redis;

```

```
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import net.oschina.j2cache.Level2Cache;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.core.RedisCallback;
import org.springframework.data.redis.core.RedisTemplate;

public class SpringRedisGenericCache implements Level2Cache
{

    /**
     * 日志
     */
    private final static Logger log =
LoggerFactory.getLogger(net.oschina.j2cache.cache.support.redis.SpringRedis
GenericCache.class);

    /**
     * 名称空间
     */
    private String namespace;

    private String region;

    private RedisTemplate<String, Serializable> redisTemplate;

    public SpringRedisGenericCache(String namespace, String region,
RedisTemplate<String, Serializable> redisTemplate)
    {
        if (region == null || region.isEmpty())
        {
            region = "_"; // 缺省region
        }
        this.namespace = namespace;
        this.redisTemplate = redisTemplate;
        this.region = getRegionName(region);
    }

    private String getRegionName(String region)
    {
        if (namespace != null && !namespace.isEmpty())
        {
            region = namespace + ":" + region;
        }
        return region;
    }

    @Override
    public void clear()
    {
        Collection<String> keys = keys();
        keys.stream().forEach(k ->
```

```
        {
            redisTemplate.delete(this.region + ":" + k);
        });
    }

    @Override
    public boolean exists(String key)
    {
        return redisTemplate.execute((RedisCallback<Boolean>) redis ->
        {
            return redis.exists(_key(key));
        });
    }

    @Override
    public void evict(String... keys)
    {
        for (String k : keys)
        {
            redisTemplate.execute((RedisCallback<Long>) redis ->
            {
                return redis.del(_key(k));
            });
        }
    }

    @Override
    public Collection<String> keys()
    {
        return redisTemplate.keys(this.region + ":*").stream().map(k ->
                k.substring(this.region.length() +
1)).collect(Collectors.toSet());
    }

    @Override
    public byte[] getBytes(String key)
    {
        return redisTemplate.execute((RedisCallback<byte[]>) redis ->
        {
            return redis.get(_key(key));
        });
    }

    @Override
    public List<byte[]> getBytes(Collection<String> keys)
    {
        return redisTemplate.execute((RedisCallback<List<byte[]>>) redis ->
        {
            byte[][] bytes = keys.stream().map(k -> _key(k)).toArray(byte[]
[]::new);
            return redis.mGet(bytes);
        });
    }

    @Override
    public void setBytes(String key, byte[] bytes, long
timeToLiveInSeconds)
    {
```

```
        if (timeToLiveInSeconds <= 0)
        {
            log.debug(String.format("Invalid timeToLiveInSeconds value : %d
, skipped it.", timeToLiveInSeconds));
            setBytes(key, bytes);
        }
        else
        {
            redisTemplate.execute((RedisCallback<List<byte[]>>) redis ->
            {
                redis.setEx(_key(key), (int) timeToLiveInSeconds, bytes);
                return null;
            });
        }
    }

    @Override
    public void setBytes(Map<String, byte[]> bytes, long
timeToLiveInSeconds)
    {
        bytes.forEach((k, v) -> setBytes(k, v, timeToLiveInSeconds));
    }

    @Override
    public void setBytes(String key, byte[] bytes)
    {
        redisTemplate.execute((RedisCallback<byte[]>) redis ->
        {
            redis.set(_key(key), bytes);
            return null;
        });
    }

    @Override
    public void setBytes(Map<String, byte[]> bytes)
    {
        bytes.forEach((k, v) -> setBytes(k, v));
    }

    private byte[] _key(String key)
    {
        byte[] k;
        try
        {
            k = (this.region + ":" + key).getBytes("utf-8");
        }
        catch (UnsupportedEncodingException e)
        {
            e.printStackTrace();
            k = (this.region + ":" + key).getBytes();
        }
        return k;
    }
}
```

第十二步：添加类SpringRedisMessageListener

```
package net.oschina.j2cache.cache.support.redis;

import net.oschina.j2cache.Command;
import net.oschina.j2cache.cluster.ClusterPolicy;
import net.oschina.j2cache.util.SerializationUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.connection.Message;
import org.springframework.data.redis.connection.MessageListener;

/**
 * spring redis 订阅消息监听
 */
public class SpringRedisMessageListener implements MessageListener
{

    /**
     * 日志记录器
     */
    private static Logger logger = LoggerFactory.getLogger(net.oschina.j2cache.cache.support.redis.SpringRedisMessageListener.class);
    /**
     * 当地命令id
     */
    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

    /**
     * 集群政策
     */
    private ClusterPolicy clusterPolicy;

    /**
     * 通道
     */
    private String channel;

    SpringRedisMessageListener(ClusterPolicy clusterPolicy, String channel)
    {
        this.clusterPolicy = clusterPolicy;
        this.channel = channel;
    }

    private boolean isLocalCommand(Command cmd)
    {
        return cmd.getSrc() == LOCAL_COMMAND_ID;
    }

    @Override
    public void onMessage(Message message, byte[] pattern)
    {
        byte[] messageChannel = message.getChannel();
```

```
        byte[] messageBody = message.getBody();
        if (messageChannel == null || messageBody == null)
        {
            return;
        }
        try
        {
            Command cmd =
Command.parse(String.valueOf(SerializationUtils.deserialize(messageBody)));
            if (cmd == null || isLocalCommand(cmd))
            {
                return;
            }

            switch (cmd.getOperator())
            {
                case Command.OPT_JOIN:
                    logger.info("Node-" + cmd.getSrc() + " joined to " +
this.channel);
                    break;
                case Command.OPT_EVICT_KEY:
                    clusterPolicy.evict(cmd.getRegion(), cmd.getKeys());
                    logger.debug("Received cache evict message, region=" +
cmd.getRegion() + ",key=" + String.join(",", cmd.getKeys()));
                    break;
                case Command.OPT_CLEAR_KEY:
                    clusterPolicy.clear(cmd.getRegion());
                    logger.debug("Received cache clear message, region=" +
cmd.getRegion());
                    break;
                case Command.OPT_QUIT:
                    logger.info("Node-" + cmd.getSrc() + " quit to " +
this.channel);
                    break;
                default:
                    logger.warn("Unknown message type = " +
cmd.getOperator());
            }
        }
        catch (Exception e)
        {
            logger.error("Failed to handle received msg", e);
        }
    }

}
```

第十三步：添加类SpringRedisProvider

```
package net.oschina.j2cache.cache.support.redis;

```

```java
import java.io.Serializable;
import java.util.Collection;
import java.util.Collections;
import java.util.Properties;
import java.util.concurrent.ConcurrentHashMap;

import net.oschina.j2cache.Cache;
import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.CacheExpiredListener;
import net.oschina.j2cache.CacheObject;
import net.oschina.j2cache.CacheProvider;
import net.oschina.j2cache.NullCache;
import net.oschina.j2cache.cache.support.util.SpringUtil;
import org.springframework.data.redis.core.RedisTemplate;

/**
 * spring redis缓存
 */
public class SpringRedisProvider implements CacheProvider
{

    /**
     * 缓存
     */
    protected ConcurrentHashMap<String, Cache> caches = new
ConcurrentHashMap<>();
    private RedisTemplate<String, Serializable> redisTemplate;
    /**
     * 配置
     */
    private net.oschina.j2cache.autoconfigure.J2CacheConfig config;
    /**
     * 名称空间
     */
    private String namespace;
    /**
     * 存储
     */
    private String storage;

    @Override
    public String name()
    {
        return "redis";
    }

    @Override
    public int level()
    {
        return CacheObject.LEVEL_2;
    }

    @Override
    public Collection<CacheChannel.Region> regions()
    {
        return Collections.emptyList();
    }

```

```
    /**
     * 建立缓存
     *
     * @param region   地区
     * @param listener 侦听器
     * @return {@link Cache}
     */
    @Override
    public Cache buildCache(String region, CacheExpiredListener listener)
    {
        if (config.getL2CacheOpen() == false)
        {
            return new NullCache();
        }
        Cache cache = caches.get(region);
        if (cache == null)
        {
            synchronized
(net.oschina.j2cache.cache.support.redis.SpringRedisProvider.class)
            {
                cache = caches.get(region);
                if (cache == null)
                {
                    if ("hash".equalsIgnoreCase(this.storage))
                    {
                        cache = new SpringRedisCache(this.namespace,
region, redisTemplate);
                    }
                    else
                    {
                        cache = new SpringRedisGenericCache(this.namespace,
region, redisTemplate);
                    }
                    caches.put(region, cache);
                }
            }
        }
        return cache;
    }

    @Override
    public Cache buildCache(String region, long timeToLiveInSeconds,
CacheExpiredListener listener)
    {
        return buildCache(region, listener);
    }

    @SuppressWarnings("unchecked")
    @Override
    public void start(Properties props)
    {
        this.namespace = props.getProperty("namespace");
        this.storage = props.getProperty("storage");
        this.config =
SpringUtil.getBean(net.oschina.j2cache.autoconfigure.J2CacheConfig.class);
        if (config.getL2CacheOpen() == false)
        {
            return;
```

```
        }
        this.redisTemplate = SpringUtil.getBean("j2CacheRedisTemplate", RedisTemplate.class);
    }

    @Override
    public void stop()
    {
        // 由spring控制
    }

}
```

第十四步：添加类SpringRedisPubSubPolicy

```
package net.oschina.j2cache.cache.support.redis;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import net.oschina.j2cache.CacheProviderHolder;
import net.oschina.j2cache.Command;
import net.oschina.j2cache.J2CacheConfig;
import net.oschina.j2cache.cache.support.util.SpringUtil;
import net.oschina.j2cache.cluster.ClusterPolicy;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.data.redis.listener.RedisMessageListenerContainer;

/**
 * 使用spring redis实现订阅功能
 */
public class SpringRedisPubSubPolicy implements ClusterPolicy
{

    /**
     * 是否是主动模式
     */
    private static boolean isActive = false;
    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
    private RedisTemplate<String, Serializable> redisTemplate;
    private net.oschina.j2cache.autoconfigure.J2CacheConfig config;
    private CacheProviderHolder holder;
    private String channel = "j2cache_channel";

    @Override
    public boolean isLocalCommand(Command cmd)
```

```
    {
        return cmd.getSrc() == LOCAL_COMMAND_ID;
    }

    @SuppressWarnings("unchecked")
    @Override
    public void connect(Properties props, CacheProviderHolder holder)
    {
        this.holder = holder;
        this.config =
SpringUtil.getBean(net.oschina.j2cache.autoconfigure.J2CacheConfig.class);
        if (config.getL2CacheOpen() == false)
        {
            return;
        }
        J2CacheConfig j2config = SpringUtil.getBean(J2CacheConfig.class);
        this.redisTemplate = SpringUtil.getBean("j2CacheRedisTemplate",
RedisTemplate.class);
        String channel_name =
j2config.getL2CacheProperties().getProperty("channel");
        if (channel_name != null && !channel_name.isEmpty())
        {
            this.channel = channel_name;
        }
        RedisMessageListenerContainer listenerContainer =
SpringUtil.getBean("j2CacheRedisMessageListenerContainer",
RedisMessageListenerContainer.class);
        String namespace =
j2config.getL2CacheProperties().getProperty("namespace");
        String database =
j2config.getL2CacheProperties().getProperty("database");
        String expired = "__keyevent@" + (database == null ||
"".equals(database) ? "0" : database) + "__:expired";
        String del = "__keyevent@" + (database == null ||
"".equals(database) ? "0" : database) + "__:del";
        List<PatternTopic> topics = new ArrayList<>();
        topics.add(new PatternTopic(expired));
        topics.add(new PatternTopic(del));

        if ("active".equals(config.getCacheCleanMode()))
        {
            isActive = true;
            //设置键值回调 需要redis支持键值回调
            ConfigureNotifyKeyspaceEventsAction action = new
ConfigureNotifyKeyspaceEventsAction();

 action.config(listenerContainer.getConnectionFactory().getConnection());
            listenerContainer.addMessageListener(new
SpringRedisActiveMessageListener(this, namespace), topics);
        }
        else if ("blend".equals(config.getCacheCleanMode()))
        {
            //设置键值回调 需要redis支持键值回调
            ConfigureNotifyKeyspaceEventsAction action = new
ConfigureNotifyKeyspaceEventsAction();

 action.config(listenerContainer.getConnectionFactory().getConnection());
```

```
            listenerContainer.addMessageListener(new
SpringRedisActiveMessageListener(this, namespace), topics);
            listenerContainer.addMessageListener(new
SpringRedisMessageListener(this, this.channel), new
PatternTopic(this.channel));
        }
        else
        {
            listenerContainer.addMessageListener(new
SpringRedisMessageListener(this, this.channel), new
PatternTopic(this.channel));
        }

    }

    /**
     * 删除本地某个缓存条目
     *
     * @param region 区域名称
     * @param keys   缓存键值
     */
    public void evict(String region, String... keys)
    {
        holder.getLevel1Cache(region).evict(keys);
    }

    /**
     * 清除本地整个缓存区域
     *
     * @param region 区域名称
     */
    public void clear(String region)
    {
        holder.getLevel1Cache(region).clear();
    }

    /**
     * 发布
     *
     * @param cmd cmd
     */
    @Override
    public void publish(Command cmd)
    {
        if (!isActive && config.getL2CacheOpen())
        {
            cmd.setSrc(LOCAL_COMMAND_ID);
            redisTemplate.convertAndSend(this.channel, cmd.json());
        }
    }

    /**
     * 断开连接
     */
    @Override
    public void disconnect()
    {
        if (!isActive && config.getL2CacheOpen())
```

```
        {
            Command cmd = new Command();
            cmd.setSrc(LOCAL_COMMAND_ID);
            cmd.setOperator(Command.OPT_QUIT);
            redisTemplate.convertAndSend(this.channel, cmd.json());
        }
    }

}
```

第十五步：添加配置类J2CacheAutoConfiguration

```
package net.oschina.j2cache.autoconfigure;

import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.J2Cache;
import net.oschina.j2cache.J2CacheBuilder;
import net.oschina.j2cache.cache.support.util.SpringJ2CacheConfigUtil;
import net.oschina.j2cache.cache.support.util.SpringUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.DependsOn;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.StandardEnvironment;

import java.io.IOException;

/**
 * 启动入口
 */
@ConditionalOnClass(J2Cache.class)
@EnableConfigurationProperties({J2CacheConfig.class})
@Configuration
@PropertySource(value = "${j2cache.config-location}", encoding = "UTF-8",
ignoreResourceNotFound = true)
public class J2CacheAutoConfiguration
{

    @Autowired
    private StandardEnvironment standardEnvironment;

    @Bean
    public net.oschina.j2cache.J2CacheConfig j2CacheConfig() throws
IOException
    {
        net.oschina.j2cache.J2CacheConfig cacheConfig =
SpringJ2CacheConfigUtil.initFromConfig(standardEnvironment);
```

```
        return cacheConfig;
    }

    @Bean
    @DependsOn({"springUtil", "j2CacheConfig"})
    public CacheChannel cacheChannel(net.oschina.j2cache.J2CacheConfig
j2CacheConfig) throws IOException
    {
        J2CacheBuilder builder = J2CacheBuilder.init(j2CacheConfig);
        return builder.getChannel();
    }

    @Bean
    public SpringUtil springUtil()
    {
        return new SpringUtil();
    }

}
```

第十六步：添加配置属性类J2CacheConfig

```
package net.oschina.j2cache.autoconfigure;

import org.springframework.boot.context.properties.ConfigurationProperties;

/**
 * 相关的配置信息
 */
@ConfigurationProperties(prefix = "j2cache")
public class J2CacheConfig
{
    private String configLocation = "/j2cache.properties";

    /**
     * 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
     */
    private Boolean openSpringCache = false;

    /**
     * 缓存清除模式,
     * <ul>
     * <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>
     * <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>
     * <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>
     * </ul>
     */
    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
```

```
    public void setAllowNullValues(boolean allowNullValues)
    {
        this.allowNullValues = allowNullValues;
    }

    public String getRedisClient()
    {
        return redisClient;
    }

    public void setRedisClient(String redisClient)
    {
        this.redisClient = redisClient;
    }

    public boolean getL2CacheOpen()
    {
        return l2CacheOpen;
    }

    public void setL2CacheOpen(boolean l2CacheOpen)
    {
        this.l2CacheOpen = l2CacheOpen;
    }

}
```

第十七步：添加配置类J2CacheSpringCacheAutoConfiguration

```
package net.oschina.j2cache.autoconfigure;

import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.J2Cache;
import net.oschina.j2cache.cache.support.J2CacheCacheManger;
import org.springframework.boot.autoconfigure.cache.CacheProperties;
import org.springframework.boot.autoconfigure.condition.ConditionalOnBean;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.List;

/**
 * 开启对spring cache支持的配置入口
 */
@Configuration
```

```
@ConditionalOnClass(J2Cache.class)
@EnableConfigurationProperties({J2CacheConfig.class, CacheProperties.class})
@ConditionalOnProperty(name = "j2cache.open-spring-cache", havingValue =
"true")
@EnableCaching
public class J2CacheSpringCacheAutoConfiguration
{

    private final CacheProperties cacheProperties;

    private final J2CacheConfig j2CacheConfig;

    J2CacheSpringCacheAutoConfiguration(CacheProperties cacheProperties,
J2CacheConfig j2CacheConfig)
    {
        this.cacheProperties = cacheProperties;
        this.j2CacheConfig = j2CacheConfig;
    }

    @Bean
    @ConditionalOnBean(CacheChannel.class)
    public J2CacheCacheManger cacheManager(CacheChannel cacheChannel)
    {
        List<String> cacheNames = cacheProperties.getCacheNames();
        J2CacheCacheManger cacheCacheManger = new
J2CacheCacheManger(cacheChannel);

 cacheCacheManger.setAllowNullValues(j2CacheConfig.isAllowNullValues());
        cacheCacheManger.setCacheNames(cacheNames);
        return cacheCacheManger;
    }


}
```

第十八步：添加配置类J2CacheSpringRedisAutoConfiguration

```
package net.oschina.j2cache.autoconfigure;

import java.io.Serializable;
import java.net.URI;
import java.net.URISyntaxException;
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import net.oschina.j2cache.cache.support.util.J2CacheSerializer;
import net.oschina.j2cache.redis.RedisUtils;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
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.l2-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 =
LoggerFactory.getLogger(net.oschina.j2cache.autoconfigure.J2CacheSpringRedi
sAutoConfiguration.class);
61
62      @SuppressWarnings("deprecation")
63      @Bean("j2CahceRedisConnectionFactory")
64      @ConditionalOnMissingBean(name = "j2CahceRedisConnectionFactory")
65      @ConditionalOnProperty(name = "j2cache.redis-client", havingValue =
"jedis", matchIfMissing = true)
66      public JedisConnectionFactory
jedisConnectionFactory(net.oschina.j2cache.J2CacheConfig j2CacheConfig)
67      {
68          Properties l2CacheProperties =
j2CacheConfig.getL2CacheProperties();
69          String hosts = l2CacheProperties.getProperty("hosts");
70          String mode = l2CacheProperties.getProperty("mode") == null ?
"null" : l2CacheProperties.getProperty("mode");
71          String clusterName = l2CacheProperties.getProperty("cluster_name");
72          String password = l2CacheProperties.getProperty("password");
73          int database = l2CacheProperties.getProperty("database") == null ?
0
74                  :
Integer.parseInt(l2CacheProperties.getProperty("database"));
75          JedisConnectionFactory connectionFactory = null;
76          JedisPoolConfig config =
RedisUtils.newPoolConfig(l2CacheProperties, null);
77          List<RedisNode> nodes = new ArrayList<>();
78          if (hosts != null && !"".equals(hosts))
79          {
80              for (String node : hosts.split(","))
81              {
82                  String[] s = node.split(":");
83                  String host = s[0];
84                  int port = (s.length > 1) ? Integer.parseInt(s[1]) : 6379;
85                  RedisNode n = new RedisNode(host, port);
86                  nodes.add(n);
87              }
88          }
89          else
90          {
91              log.error("j2cache中的redis配置缺少hosts!！");
92              throw new IllegalArgumentException("j2cache中的redis配置缺少
hosts");
93          }
94
95          RedisPassword paw = RedisPassword.none();
96          if (!StringUtils.isEmpty(password))
97          {
98              paw = RedisPassword.of(password);
99          }
100
101         switch (mode)
102         {
103             case "sentinel":
```

```
                RedisSentinelConfiguration sentinel = new
RedisSentinelConfiguration();
                sentinel.setDatabase(database);
                sentinel.setPassword(paw);
                sentinel.setMaster(clusterName);
                sentinel.setSentinels(nodes);
                connectionFactory = new JedisConnectionFactory(sentinel,
config);
                break;
            case "cluster":
                RedisClusterConfiguration cluster = new
RedisClusterConfiguration();
                cluster.setClusterNodes(nodes);
                cluster.setMaxRedirects(MAX_ATTEMPTS);
                cluster.setPassword(paw);
                connectionFactory = new JedisConnectionFactory(cluster,
config);
                break;
            case "sharded":
                try
                {
                    for (String node : hosts.split(","))
                    {
                        connectionFactory = new JedisConnectionFactory(new
JedisShardInfo(new URI(node)));
                        connectionFactory.setPoolConfig(config);
                        log.warn("Jedis mode [sharded] not recommended for
use!!");
                        break;
                    }
                }
                catch (URISyntaxException e)
                {
                    throw new JedisConnectionException(e);
                }
                break;
            default:
                for (RedisNode node : nodes)
                {
                    String host = node.getHost();
                    int port = node.getPort();
                    RedisStandaloneConfiguration single = new
RedisStandaloneConfiguration(host, port);
                    single.setDatabase(database);
                    single.setPassword(paw);
                    JedisClientConfigurationBuilder clientConfiguration =
JedisClientConfiguration.builder();
                    clientConfiguration.usePooling().poolConfig(config);

 clientConfiguration.connectTimeout(Duration.ofMillis(CONNECT_TIMEOUT));
                    connectionFactory = new JedisConnectionFactory(single,
clientConfiguration.build());
                    break;
                }
                if (!"single".equalsIgnoreCase(mode))
                {
                    log.warn("Redis mode [" + mode + "] not defined. Using
'single'.");
```

```
                }
                break;
        }
        return connectionFactory;
    }

    @Primary
    @Bean("j2CahceRedisConnectionFactory")
    @ConditionalOnMissingBean(name = "j2CahceRedisConnectionFactory")
    @ConditionalOnProperty(name = "j2cache.redis-client", havingValue =
"lettuce")
    public LettuceConnectionFactory
lettuceConnectionFactory(net.oschina.j2cache.J2CacheConfig j2CacheConfig)
    {
        Properties l2CacheProperties =
j2CacheConfig.getL2CacheProperties();
        String hosts = l2CacheProperties.getProperty("hosts");
        String mode = l2CacheProperties.getProperty("mode") == null ?
"null" : l2CacheProperties.getProperty("mode");
        String clusterName = l2CacheProperties.getProperty("cluster_name");
        String password = l2CacheProperties.getProperty("password");
        int database = l2CacheProperties.getProperty("database") == null ?
0
                :
Integer.parseInt(l2CacheProperties.getProperty("database"));
        LettuceConnectionFactory connectionFactory = null;
        LettucePoolingClientConfigurationBuilder config =
LettucePoolingClientConfiguration.builder();
        config.commandTimeout(Duration.ofMillis(CONNECT_TIMEOUT));
        config.poolConfig(getGenericRedisPool(l2CacheProperties, null));
        List<RedisNode> nodes = new ArrayList<>();
        if (hosts != null && !"".equals(hosts))
        {
            for (String node : hosts.split(","))
            {
                String[] s = node.split(":");
                String host = s[0];
                int port = (s.length > 1) ? Integer.parseInt(s[1]) : 6379;
                RedisNode n = new RedisNode(host, port);
                nodes.add(n);
            }
        }
        else
        {
            log.error("j2cache中的redis配置缺少hosts!!");
            throw new IllegalArgumentException();
        }
        RedisPassword paw = RedisPassword.none();
        if (!StringUtils.isEmpty(password))
        {
            paw = RedisPassword.of(password);
        }
        switch (mode)
        {
            case "sentinel":
                RedisSentinelConfiguration sentinel = new
RedisSentinelConfiguration();
                sentinel.setDatabase(database);
```

```
                sentinel.setPassword(paw);
                sentinel.setMaster(clusterName);
                sentinel.setSentinels(nodes);
                connectionFactory = new LettuceConnectionFactory(sentinel,
config.build());
                break;
            case "cluster":
                RedisClusterConfiguration cluster = new
RedisClusterConfiguration();
                cluster.setClusterNodes(nodes);
                cluster.setMaxRedirects(MAX_ATTEMPTS);
                cluster.setPassword(paw);
                connectionFactory = new LettuceConnectionFactory(cluster,
config.build());
                break;
            case "sharded":
                throw new IllegalArgumentException("Lettuce not support use
mode [sharded]!!");
            default:
                for (RedisNode node : nodes)
                {
                    String host = node.getHost();
                    int port = node.getPort();
                    RedisStandaloneConfiguration single = new
RedisStandaloneConfiguration(host, port);
                    single.setDatabase(database);
                    single.setPassword(paw);
                    connectionFactory = new
LettuceConnectionFactory(single, config.build());
                    break;
                }
                if (!"single".equalsIgnoreCase(mode))
                {
                    log.warn("Redis mode [" + mode + "] not defined. Using
'single'.");
                }
                break;
        }
        return connectionFactory;
    }

    @Bean("j2CacheRedisTemplate")
    public RedisTemplate<String, Serializable> j2CacheRedisTemplate(
            @Qualifier("j2CahceRedisConnectionFactory")
RedisConnectionFactory j2CahceRedisConnectionFactory,
            @Qualifier("j2CacheValueSerializer") RedisSerializer<Object>
j2CacheSerializer)
    {
        RedisTemplate<String, Serializable> template = new
RedisTemplate<String, Serializable>();
        template.setKeySerializer(new StringRedisSerializer());
        template.setHashKeySerializer(new StringRedisSerializer());
        template.setDefaultSerializer(j2CacheSerializer);
        template.setConnectionFactory(j2CahceRedisConnectionFactory);
        return template;
    }

    @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")
RedisConnectionFactory j2CahceRedisConnectionFactory)
258     {
259         RedisMessageListenerContainer container = new
RedisMessageListenerContainer();
260         container.setConnectionFactory(j2CahceRedisConnectionFactory);
261         return container;
262     }
263 
264     private GenericObjectPoolConfig getGenericRedisPool(Properties props,
String prefix)
265     {
266         GenericObjectPoolConfig cfg = new GenericObjectPoolConfig();
267         cfg.setMaxTotal(Integer.valueOf((String)
props.getOrDefault(key(prefix, "maxTotal"), "-1")));
268         cfg.setMaxIdle(Integer.valueOf((String)
props.getOrDefault(key(prefix, "maxIdle"), "100")));
269         cfg.setMaxWaitMillis(Integer.valueOf((String)
props.getOrDefault(key(prefix, "maxWaitMillis"), "100")));
270         cfg.setMinEvictableIdleTimeMillis(
271                 Integer.valueOf((String) props.getOrDefault(key(prefix,
"minEvictableIdleTimeMillis"), "864000000")));
272         cfg.setMinIdle(Integer.valueOf((String)
props.getOrDefault(key(prefix, "minIdle"), "10")));
273         cfg.setNumTestsPerEvictionRun(
274                 Integer.valueOf((String) props.getOrDefault(key(prefix,
"numTestsPerEvictionRun"), "10")));
275         cfg.setLifo(Boolean.valueOf(props.getProperty(key(prefix, "lifo"),
"false")));
276         cfg.setSoftMinEvictableIdleTimeMillis(
277                 Integer.valueOf((String) props.getOrDefault(key(prefix,
"softMinEvictableIdleTimeMillis"), "10")));
278         cfg.setTestOnBorrow(Boolean.valueOf(props.getProperty(key(prefix,
"testOnBorrow"), "true")));
279         cfg.setTestOnReturn(Boolean.valueOf(props.getProperty(key(prefix,
"testOnReturn"), "false")));
280         cfg.setTestWhileIdle(Boolean.valueOf(props.getProperty(key(prefix,
"testWhileIdle"), "true")));
281         cfg.setTimeBetweenEvictionRunsMillis(
282                 Integer.valueOf((String) props.getOrDefault(key(prefix,
"timeBetweenEvictionRunsMillis"), "300000")));
283 
 cfg.setBlockWhenExhausted(Boolean.valueOf(props.getProperty(key(prefix,
"blockWhenExhausted"), "false")));
284         return cfg;
285     }
286 
287     private String key(String prefix, String key)
288     {
289         return (prefix == null) ? key : prefix + "." + key;
```

```
    }
}
```

第十九步：添加配置类CacheConfig

```
package mao.tools_j2cache.config;

import org.springframework.cache.annotation.CachingConfigurerSupport;
import org.springframework.cache.interceptor.KeyGenerator;

/**
 * Project name(项目名称)：j2cache_spring_boot_starter_demo
 * Package(包名): mao.tools_j2cache.config
 * Class(类名): CacheConfig
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 21:22
 * Version(版本): 1.0
 * Description(描述)：  覆盖 SpringCache 相关配置
 */

public class CacheConfig extends CachingConfigurerSupport
{
    /**
     * 解决注解：Cacheable 没有指定key时，会将key生成为 ${value}:SimpleKey []
     * eg： @Cacheable(value = "pinda") ->  pinda:SimpleKey []
     *
     * @return {@link KeyGenerator}
     */
    @Override
    public KeyGenerator keyGenerator()
    {
        return (target, method, objects) ->
        {
            /*StringBuilder sb = new StringBuilder();
            sb.append(target.getClass().getName());
            sb.append(StrPool.COLON);
            sb.append(method.getName());
            for (Object obj : objects) {
                if (obj != null) {
                    sb.append(StrPool.COLON);
                    sb.append(obj.toString());
                }
            }
            return sb.toString();*/
            return "";
        };
    }
```

```
}
```

第二十步：添加实体类RedisData

```
package mao.tools_j2cache.entity;

import java.time.LocalDateTime;

/**
 * Project name(项目名称)：j2cache_spring_boot_starter_demo
 * Package(包名): mao.tools_j2cache.entity
 * Class(类名): RedisData
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 22:29
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class RedisData
{
    /**
     * 数据
     */
    private Object data;

    /**
     * 过期时间
     */
    private LocalDateTime expireTime;

    /**
     * Instantiates a new Redis data.
     */
    public RedisData()
    {

    }

    /**
     * Instantiates a new Redis data.
     *
     * @param data       the data
     * @param expireTime the expire time
     */
    public RedisData(Object data, LocalDateTime expireTime)
    {
        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  }
```

第二十一步：添加配置类RedissonConfig

```
package mao.tools_j2cache.config;

import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.config.ClusterServersConfig;
import org.redisson.config.Config;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.annotation.PostConstruct;

/**
 * Project name(项目名称)：j2cache_spring_boot_starter_demo
 * Package(包名): mao.tools_j2cache.config
 * Class(类名): RedissonConfig
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 22:47
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@Configuration
public class RedissonConfig
{
    /**
     * 日志
     */
    private static final Logger log =
LoggerFactory.getLogger(RedissonConfig.class);

    /**
     * Redisson配置
     *
     * @return RedissonClient
     */
    @Bean
    public RedissonClient redissonClient(@Value("${redis.hosts}") String
hosts, @Value("${redis.password}") String password)
    {
        if (!hosts.contains(","))
        {
            //没有逗号，单机模式
            log.info("单机模式redis:" + hosts);
            //配置类
            Config config = new Config();
            //添加redis地址，用config.useClusterServers()添加集群地址
```

```
            config.useSingleServer().setAddress("redis://" +
hosts).setPassword(password);
            //创建客户端
            return Redisson.create(config);
        }
        else
        {
            //集群
            log.info("集群模式redis:" + hosts);
            String[] hosts_ = hosts.split(",");
            //配置类
            Config config = new Config();
            //添加redis地址，用config.useClusterServers()添加集群地址
            ClusterServersConfig clusterServersConfig =
config.useClusterServers();
            for (String host : hosts_)
            {
                clusterServersConfig.addNodeAddress("redis://" + host);
            }
            //config.useSingleServer().setAddress("redis://" +
hosts).setPassword(password);
            //创建客户端
            return Redisson.create(config);
        }

    }

    @PostConstruct
    public void init()
    {
        log.info("初始化 RedissonConfig");
    }
}
```

第二十二步：添加工具类RedisUtils

```
package mao.tools_j2cache.utils;

import cn.hutool.core.util.StrUtil;
import cn.hutool.json.JSONObject;
import cn.hutool.json.JSONUtil;
import mao.tools_j2cache.entity.RedisData;
import org.redisson.api.RLock;
import org.redisson.api.RedissonClient;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.core.StringRedisTemplate;


import javax.annotation.Resource;
import java.time.LocalDateTime;
```

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.function.Function;


/**
 * Project name(项目名称)：j2cache_spring_boot_starter_demo
 * Package(包名): mao.tools_j2cache.utils
 * Class(类名): RedisUtils
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 22:25
 * Version(版本): 1.0
 * Description(描述)： 缓存工具类
 */


public class RedisUtils
{

    /**
     * 日志
     */
    private static final Logger log =
LoggerFactory.getLogger(RedisUtils.class);

    @Resource
    private StringRedisTemplate stringRedisTemplate;

    @Resource
    private RedissonClient redissonClient;

    //线程池
    private static final ExecutorService CACHE_REBUILD_EXECUTOR =
Executors.newFixedThreadPool(10);

    /**
     * 向redis里添加数据
     *
     * @param redisKey   redis的key
     * @param value      数据
     * @param expireTime 过期时间
     * @param timeUnit   时间单位
     */
    public void set(String redisKey, Object value, Long expireTime,
TimeUnit timeUnit)
    {
        stringRedisTemplate.opsForValue().set(redisKey,
JSONUtil.toJsonStr(value), expireTime, timeUnit);
    }


    /**
     * 向redis里添加数据 设置逻辑过期
     *
```

```
     * @param redisKey   redis的key
     * @param value      数据
     * @param expireTime 过期时间
     * @param timeUnit   时间单位
     */
    public void setWithLogicalExpire(String redisKey, Object value, Long expireTime, TimeUnit timeUnit)
    {
        RedisData redisData = new RedisData();
        //添加数据
        redisData.setData(value);
        //设置过期时间
        redisData.setExpireTime(LocalDateTime.now().plusSeconds(timeUnit.toSeconds(expireTime)));
        //放入redis
        stringRedisTemplate.opsForValue().set(redisKey, JSONUtil.toJsonStr(redisData));
    }


    /**
     * 查询数据，有缓存，解决缓存穿透问题，未解决缓存雪崩问题
     *
     * @param <R>         返回值的类型
     * @param <ID>        id的类型
     * @param keyPrefix   redisKey的前缀
     * @param id          id
     * @param type        返回值的类型
     * @param dbFallback 查询数据库的函数
     * @param expireTime 过期时间
     * @param timeUnit   时间单位
     * @return 泛型R r
     */
    public <R, ID> R queryWithPassThrough(String keyPrefix, ID id, Class<R> type,
                                          Function<ID, R> dbFallback, Long expireTime, TimeUnit timeUnit)
    {
        //获得前缀
        String redisKey = keyPrefix + id;
        //查询redis
        String json = stringRedisTemplate.opsForValue().get(redisKey);
        //判断是否为空
        if (StrUtil.isNotBlank(json))
        {
            //不为空，返回
            return JSONUtil.toBean(json, type);
        }
        //判断是否为空串
        if (json != null)
        {
            //空串
            return null;
        }
        //null
        //查数据库
        R r = dbFallback.apply(id);
```

```
        //判断
        if (r == null)
        {
            //数据库也为空，缓存空值
            this.set(redisKey, "", expireTime, timeUnit);
            return null;
        }
        //数据库存在，写入redis
        this.set(redisKey, r, expireTime, timeUnit);
        //返回
        return r;
    }

    /**
     * 查询数据，有缓存，解决缓存穿透问题，解决缓存雪崩问题
     *
     * @param <R>                                返回值的类型
     * @param <ID>                               id的类型
     * @param keyPrefix                          redisKey的前缀
     * @param id                                 id
     * @param type                               返回值的类型
     * @param dbFallback                         查询数据库的函数
     * @param expireTime                         过期时间
     * @param timeUnit                           时间单位
     * @param maxTimeSecondsByCacheAvalanche this.set(redisKey, r,
     *
timeUnit.toSeconds(expireTime)+getIntRandom(0,maxTimeSecondsByCacheAvalanch
e),
     *                                           TimeUnit.SECONDS);
     * @return 泛型R r
     */
    public <R, ID> R queryWithPassThroughAndCacheAvalanche(String
keyPrefix, ID id, Class<R> type,
                                                           Function<ID, R>
dbFallback, Long expireTime, TimeUnit timeUnit,
                                                           Integer
maxTimeSecondsByCacheAvalanche)
    {
        //获得前缀
        String redisKey = keyPrefix + id;
        //查询redis
        String json = stringRedisTemplate.opsForValue().get(redisKey);
        //判断是否为空
        if (StrUtil.isNotBlank(json))
        {
            //不为空，返回
            return JSONUtil.toBean(json, type);
        }
        //判断是否为空串
        if (json != null)
        {
            //空串
            return null;
        }
        //null
        //查数据库
        R r = dbFallback.apply(id);
        //判断
```

```
        if (r == null)
        {
            //数据库也为空，缓存空值
            this.set(redisKey, "",
                    timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
                    TimeUnit.SECONDS);
            return null;
        }
        //数据库存在，写入redis
        this.set(redisKey, r,
                timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
                TimeUnit.SECONDS);
        //返回
        return r;
    }

    /**
     * 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
     *
     * @param <R>                                返回值的类型
     * @param <ID>                               id的类型
     * @param keyPrefix                          redisKey的前缀
     * @param lockKeyPrefix                      锁的前缀
     * @param id                                 id
     * @param type                               返回值的类型
     * @param dbFallback                         查询数据库的函数
     * @param expireTime                         过期时间
     * @param timeUnit                           时间单位
     * @param maxTimeSecondsByCacheAvalanche this.set(redisKey, r,
     *
timeUnit.toSeconds(expireTime)+getIntRandom(0,maxTimeSecondsByCacheAvalanch
e),                                          TimeUnit.SECONDS);
     * @return 泛型R r
     */
    public <R, ID> R query(String keyPrefix, String lockKeyPrefix, ID id,
Class<R> type,
                           Function<ID, R> dbFallback, Long expireTime,
TimeUnit timeUnit,
                           Integer maxTimeSecondsByCacheAvalanche)
    {
        //获取redisKey
        String redisKey = keyPrefix + id;
        //log.debug("查询：" + redisKey);
        //从redis中查询信息，根据id
        String json = stringRedisTemplate.opsForValue().get(redisKey);
        //判断取出的数据是否为空
        if (StrUtil.isNotBlank(json))
        {
            //log.debug(redisKey + " 缓存命中");
            //不是空，redis里有，返回
            return JSONUtil.toBean(json, type);
        }
        //是空串，不是null，返回
        if (json != null)
        {
            return null;
```

```
        }
        //锁的key
        String lockKey = lockKeyPrefix + id;

        R r = null;
        LockInfo lockInfo = null;
        try
        {
            //log.debug(redisKey + " 缓存未命中，尝试获取锁");
            //获取互斥锁
            lockInfo = tryLock(lockKey);
            //判断锁是否获取成功
            if (!lockInfo.isSuccess())
            {
                //没有获取到锁
                //200毫秒后再次获取
                Thread.sleep(200);
                //递归调用
                return query(keyPrefix, lockKeyPrefix, id, type,
dbFallback,
                        expireTime, timeUnit,
maxTimeSecondsByCacheAvalanche);
            }
            //得到了锁
            //从redis中查询信息，根据id
            json = stringRedisTemplate.opsForValue().get(redisKey);
            //判断取出的数据是否为空
            if (StrUtil.isNotBlank(json))
            {
                //log.debug(redisKey + " 获取分布式锁后，缓存命中");
                //不是空，redis里有，返回
                return JSONUtil.toBean(json, type);
            }

            //null，查数据库
            log.debug(redisKey + " 获取分布式锁后，缓存未命中，查询数据库");
            r = dbFallback.apply(id);
            //判断数据库里的信息是否为空
            if (r == null)
            {
                //数据库也为空，缓存空值
                this.set(redisKey, "",
                        timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
                        TimeUnit.SECONDS);
                return null;
            }
            //存在，回写到redis里，设置随机的过期时间
            this.set(redisKey, r,
                    timeUnit.toSeconds(expireTime) + getIntRandom(0,
maxTimeSecondsByCacheAvalanche),
                    TimeUnit.SECONDS);
        }
        catch (InterruptedException e)
        {
            throw new RuntimeException(e);
        }
        finally
```

```
        {
            //释放锁
            if (lockInfo != null)
            {
                this.unlock(lockInfo);
            }
        }
        //返回数据
        return r;
    }

    /**
     * 更新数据
     *
     * @param <T>         要更新的对象的泛型
     * @param <ID>        主键的类型
     * @param id          要更新的主键
     * @param data        要更新的对象
     * @param keyPrefix   redis的key前缀
     * @param dbFallback 更新数据库的函数，返回值要为Boolean类型
     * @return boolean boolean
     */
    public <T, ID> boolean update(ID id, T data, String keyPrefix,
Function<T, Boolean> dbFallback)
    {
        //判断是否为空
        if (id == null)
        {
            return false;
        }
        //不为空
        //先更新数据库
        boolean b = dbFallback.apply(data);
        //更新失败，返回
        if (!b)
        {
            return false;
        }
        //更新没有失败
        //删除redis里的数据，下一次查询时自动添加进redis
        //redisKey
        String redisKey = keyPrefix + id;
        stringRedisTemplate.delete(redisKey);
        log.debug("更新：" + redisKey);
        //返回响应
        return true;
    }

    /**
     * Query with logical expire r.
     *
     * @param <R>            返回值的类型
     * @param <ID>           id的类型
     * @param keyPrefix      redisKey的前缀
     * @param lockKeyPrefix 锁的前缀
     * @param id             id
     * @param type           返回值的类型
     * @param dbFallback     查询数据库的函数
```

```
     * @param time           过期时间
     * @param timeUnit       时间单位
     * @return 泛型R r
     */
    public <R, ID> R queryWithLogicalExpire(String keyPrefix, String lockKeyPrefix, ID id, Class<R> type,
                                            Function<ID, R> dbFallback, Long time, TimeUnit timeUnit)
    {
        //获得前缀
        String redisKey = keyPrefix + id;
        //查询redis
        String json = stringRedisTemplate.opsForValue().get(redisKey);
        //判断是否为空
        if (StrUtil.isBlank(json))
        {
            //空，返回
            return null;
        }
        //不为空
        //json 反序列化为对象
        RedisData redisData = JSONUtil.toBean(json, RedisData.class);
        //获得过期时间
        LocalDateTime expireTime = redisData.getExpireTime();
        //获取数据
        R r = JSONUtil.toBean((JSONObject) redisData.getData(), type);
        //判断是否过期
        if (expireTime.isAfter(LocalDateTime.now()))
        {
            //未过期，返回
            return r;
        }
        //过期，缓存重建
        //获取互斥锁
        String lockKey = lockKeyPrefix + id;
        LockInfo lockInfo = tryLock(lockKey);
        if (lockInfo.isSuccess())
        {
            //获取锁成功
            // 开辟独立线程
            CACHE_REBUILD_EXECUTOR.submit(new Runnable()
            {
                @Override
                public void run()
                {
                    try
                    {
                        R r1 = dbFallback.apply(id);
                        setWithLogicalExpire(redisKey, r1, time, timeUnit);
                    }
                    catch (Exception e)
                    {
                        throw new RuntimeException(e);
                    }
                    finally
                    {
                        //释放锁
                        unlock(lockInfo);
```

```
                    }
                }
            });
        }
        //没有获取到锁，使用旧数据返回
        return r;
    }


    /**
     * 获取锁
     *
     * @param key redisKey
     * @return {@link LockInfo}
     */
    private LockInfo tryLock(String key)
    {
        // 获取锁（可重入），指定锁的名称
        RLock lock = redissonClient.getLock(key);
        try
        {
            // 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），锁自动释放时间，时间单位
            LockInfo lockInfo = new LockInfo();
            lockInfo.setLock(lock);
            lockInfo.setSuccess(lock.tryLock(1, 10, TimeUnit.SECONDS));
            //log.debug("尝试获取分布式锁：" + lockInfo.isSuccess());
            return lockInfo;
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
            LockInfo lockInfo = new LockInfo();
            lockInfo.setLock(lock);
            lockInfo.setSuccess(false);
            return lockInfo;
        }
    }

    /**
     * 释放锁
     *
     * @param lockInfo 锁信息
     */
    private void unlock(LockInfo lockInfo)
    {
        if (lockInfo.getLock().isHeldByCurrentThread())
        {
            //log.debug("尝试释放分布式锁");
            lockInfo.lock.unlock();
        }
    }


    /**
     * 获取一个随机数，区间包含min和max
     *
     * @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             *
```

```
         * @param lock the lock
         */
        public void setLock(RLock lock)
        {
            this.lock = lock;
        }
    }
}
```

第二十三步：添加配置类RedisUtilsConfig

```
package mao.tools_j2cache.config;

import mao.tools_j2cache.utils.RedisUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

import javax.annotation.PostConstruct;

/**
 * Project name(项目名称)：j2cache_spring_boot_starter_demo
 * Package(包名): mao.tools_j2cache.config
 * Class(类名): RedisUtilsConfig
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 23:30
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@Configuration
@ConditionalOnClass(RedisUtils.class)
@Import(RedisUtils.class)
public class RedisUtilsConfig
{
    /**
     * 日志
     */
    private static final Logger log =
LoggerFactory.getLogger(RedisUtilsConfig.class);

    @PostConstruct
    public void init()
    {
        log.info("初始化 RedisUtilsConfig");
```

```
    }
}
```

第二十四步：编写spring.factories

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
    mao.tools_j2cache.config.CacheConfig,\
    net.oschina.j2cache.autoconfigure.J2CacheAutoConfiguration,\
    net.oschina.j2cache.autoconfigure.J2CacheSpringCacheAutoConfiguration,\
    net.oschina.j2cache.autoconfigure.J2CacheSpringRedisAutoConfiguration,\
    mao.tools_j2cache.config.RedissonConfig,\
    mao.tools_j2cache.config.RedisUtilsConfig
```

使用starter

第一步：添加tools-j2cache的依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>j2cache_spring_boot_starter_demo</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <artifactId>use-starter</artifactId>
    <name>use-starter</name>
    <description>use-starter</description>

    <properties>

    </properties>
```

```
    <dependencies>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>mao</groupId>
            <artifactId>tools-j2cache</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第二步：编写配置文件application.yml

```
# 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）

def:
  redis:
    ip: 127.0.0.1
    port: 6379
    password: 123456

# redis 通用配置
spring:
  cache:
    type: GENERIC

j2cache:
```

```
  # 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）
  # configLocation: /j2cache.properties
  # 是否开启 SpringCache 支持
  open-spring-cache: true
  # 清除缓存的模式
  #  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
  #  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
  #  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
  cache-clean-mode: passive
  # 是否允许存放null 值
  allow-null-values: true
  # redis 客户端  (可选值： jedis  lettuce)
  redis-client: lettuce
  # 是否开启二级缓存  开发环境可以关闭
  l2-cache-open: true

  #以下来自 j2cache.properties
  # 缓存广播方式：
  # jgroups -> 使用jgroups的多播
  # redis -> 使用redis发布/订阅机制(使用jedis)
  # lettuce -> 使用redis发布/订阅机制(使用lettuce，推荐)
  # rabbitmq -> 使用 RabbitMQ 发布/消费 机制
  # rocketmq -> 使用 RocketMQ 发布/消费 机制
  # none -> 不要通知集群中的其他节点
  # xx.xxxx.xxxx.Xxxxx 实现net.oschina.j2cache.cluster.ClusterPolicy的您自己的缓存广播策略类名
  broadcast: net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
  # 1级缓存提供商类，可选值：
  # none -> 禁用此级别缓存
  # ehcache -> 使用 ehcache2 作为1级缓存
  # ehcache3 -> 使用 ehcache3 作为1级缓存
  # caffeine -> 使用 caffeine 作为1级缓存 (仅作用于内存)
  L1:
    provider_class: caffeine
  # 2级缓存提供商类，可选值：
  # redis -> 使用 redis 作为2级缓存 (使用 jedis)
  # lettuce -> 使用 redis 作为2级缓存 (使用 lettuce)
  # 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。
  # memcached -> 使用 memcached 作为2级缓存 (使用 xmemcached),
  # [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 )
  L2:
    provider_class: net.oschina.j2cache.cache.support.redis.SpringRedisProvider
    config_section: lettuce
  # 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)
  # 注意:redis哈希模式(redis.storage = hash 和 lettuce.storage = hash)不支持此功能
  sync_ttl_to_redis: true
  # 是否默认缓存空对象(默认为false)
  default_cache_null_object: false
  # 缓存序列化提供者 ，  可选值：
```

```
  # fst -> 使用 fast 序列化 (推荐)                缺点：增删改字段后反序列化会报错
  # 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！
  # json -> 使用 fst's json 序列化 (测试中)       缺点：不支持LocalDateTime
  # 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...
  # 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
  # xxx.xxx.xxxx.Xxx -> [自定义序列化类]
  serialization: json

# j2cache.serialization=json 时可用
#json:
#  map.person: net.oschina.j2cache.demo.Person

# 广播相关配置： jgroups 配置 （当 j2cache.broadcast=jgroups 时，才需要配置）
jgroups:
  # 网络配置文件路径 （相对于/resources 目录）
  configXml: /network.xml
  # 广播渠道名称
  channel:
    name: j2cache

# 广播相关配置： rabbitmq 配置 （当 j2cache.broadcast=rabbitmq 时，才需要配置）
rabbitmq:
  exchange: j2cache
  host: localhost
  port: 5672
  username: guest
  password: guest

# 广播相关配置： rocketmq 配置 （当 j2cache.broadcast=rocketmq 时，才需要配置）
rocketmq:
  name: j2cache
  topic: j2cache
  # 使用;分割多台主机
  hosts: 127.0.0.1:9876

# 1级相关缓存配置： （当 j2cache.L1.config_section=ehcache 时，才需要配置）
ehcache:
  configXml: /ehcache.xml

# 1级相关缓存配置： （当 j2cache.L1.config_section=ehcache3 时，才需要配置）
ehcache3:
  configXml: /ehcache.xml
  defaultHeapSize: 1000

# 1级相关缓存配置： （当 j2cache.L1.config_section=caffeine 时，才需要配置）
caffeine:
  # properties 和 region.[name] 任选一种方式配置
  properties: /j2cache/caffeine.properties   # 这个配置文件需要放在项目中
  #region.[name]: size, xxxx[s|m|h|d]


# 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）
```

```
# 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）
redis:
  # Redis 集群模式
  # single -> 单 redis 服务
  # sentinel -> 主从 服务
  # cluster -> 集群 服务 (数据库配置无效，使用 database = 0）
  # 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）
  mode: single
  # redis storage mode (generic|hash)
  storage: generic
  # redis发布/订阅频道名称
  channel: j2cache
  # redis发布/订阅服务器(该值为空时，使用redis.host)
  channel.host:
  # 集群名: 仅用于分片
  cluster_name: j2cache
  # redis缓存命名空间可选，默认[空]
  namespace:
  hosts: ${def.redis.ip}:${def.redis.port}
  timeout: 2000
  password: ${def.redis.password}
  database: 0
  maxTotal: 100
  maxIdle: 10
  maxWaitMillis: 5000
  minEvictableIdleTimeMillis: 60000
  minIdle: 1
  numTestsPerEvictionRun: 10
  lifo: false
  softMinEvictableIdleTimeMillis: 10
  testOnBorrow: true
  testOnReturn: false
  testWhileIdle: true
  timeBetweenEvictionRunsMillis: 300000
  blockWhenExhausted: false
  jmxEnabled: false

# 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）
# 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）
lettuce:
  mode: single
  namespace:
  storage: generic
  channel: j2cache
  scheme: redis
  hosts: ${def.redis.ip}:${def.redis.port}
  password: ${def.redis.password}
  database: 0
  sentinelMasterId:
  maxTotal: 100
  maxIdle: 10
  minIdle: 10
```

```yaml
  timeout: 10000

# 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）
# 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）
memcached:
  servers: 127.0.0.1:11211
  username:
  password:
  connectionPoolSize: 10
  connectTimeout: 1000
  failureMode: false
  healSessionInterval: 1000
  maxQueuedNoReplyOperations: 100
  opTimeout: 100
  sanitizeKeys: false









# 设置日志级别，root表示根节点，即整体应用日志级别
logging:
  # 日志输出到文件的文件名
  file:
    name: server.log
  # 字符集
  charset:
    file: UTF-8
  # 分文件
  logback:
    rollingpolicy:
      #最大文件大小
      max-file-size: 16KB
      # 文件格式
      file-name-pattern: logs/server_log/%d{yyyy/MM月/dd日/}%i.log
  # 设置日志组
  group:
    # 自定义组名，设置当前组中所包含的包
    mao_pro: mao
  level:
    root: info
    # 为对应组设置日志级别
    mao_pro: debug
    # 日志输出格式
  # pattern:
  # console: "%d %clr(%p) --- [%16t] %clr(%-40.40c){cyan} : %m %n"
```

第三步：编写实体类Student

```java
package mao.use_starter.entity;

/**
 * Project name(项目名称)：j2cache_spring_boot_starter_demo
 * Package(包名): mao.use_starter.entity
 * Class(类名): Student
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 23:51
 * Version(版本): 1.0
 * Description(描述)： 无
 */


public class Student
{
    private Long id;
    private String name;

    /**
     * Instantiates a new Student.
     */
    public Student()
    {

    }

    /**
     * Instantiates a new Student.
     *
     * @param id   the id
     * @param name the name
     */
    public Student(Long id, String name)
    {
        this.id = id;
        this.name = name;
    }

    /**
     * Gets id.
     *
     * @return the id
     */
    public Long getId()
    {
        return id;
    }

    /**
     * Sets id.
```

```
     *
     * @param id the id
     */
    public void setId(Long id)
    {
        this.id = id;
    }

    /**
     * Gets name.
     *
     * @return the name
     */
    public String getName()
    {
        return name;
    }

    /**
     * Sets name.
     *
     * @param name the name
     */
    public void setName(String name)
    {
        this.name = name;
    }
}
```

第四步：编写TestController

```
package mao.use_starter.controller;

import mao.tools_j2cache.utils.RedisUtils;
import mao.use_starter.entity.Student;
import net.oschina.j2cache.CacheChannel;
import net.oschina.j2cache.CacheObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;
import java.util.function.Function;

/**
 * Project name(项目名称)：j2cache_demo
 * Package(包名): mao.j2cache_demo.controller
```

```
 * Class(类名): TestController
 * Author(作者）: mao
 * Author QQ: 1296193245
 * GitHub: https://github.com/maomao124/
 * Date(创建日期)： 2022/11/5
 * Time(创建时间)： 13:22
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
public class TestController
{

    private static final Logger log =
LoggerFactory.getLogger(TestController.class);

    @Autowired
    private CacheChannel cacheChannel;

    private final String key = "myKey";
    private final String region = "rx";


    @GetMapping("/getInfos")
    public List<String> getInfos()
    {
        CacheObject cacheObject = cacheChannel.get(region, key);
        if (cacheObject.getValue() == null)
        {
            log.info("查询数据库");
            //缓存中没有找到，查询数据库获得
            List<String> data = new ArrayList<>();
            data.add("info1");
            data.add("info2");
            try
            {
                Thread.sleep(9);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            //放入缓存
            cacheChannel.set(region, key, data);
            return data;
        }
        return (List<String>) cacheObject.getValue();
    }

    private String cache = null;

    @GetMapping("/getInfos2")
    public String getInfos2()
    {
        if (cache == null)
        {
            log.info("查询数据库2");
```

```
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            cache = "hello";
        }
        else
        {
            return cache;
        }
        return cache;
    }


    @GetMapping("/getInfos3")
    public List<String> getInfos3()
    {
        CacheObject cacheObject = cacheChannel.get(region, key);
        if (cacheObject.getValue() == null)
        {
            log.info("查询数据库3");
            //缓存中没有找到，查询数据库获得
            try
            {
                Thread.sleep(9);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            //放入缓存
            cacheChannel.set(region, key, null);
            return null;
        }
        return null;
    }

    /**
     * 清理指定缓存
     *
     * @return {@link String}
     */
    @GetMapping("/evict")
    public String evict()
    {
        cacheChannel.evict(region, key);
        return "evict success";
    }

    /**
     * 检测存在哪级缓存
     *
     * @return {@link String}
     */
```

```
    @GetMapping("/check")
    public String check()
    {
        int check = cacheChannel.check(region, key);
        return "level:" + check;
    }

    /**
     * 检测缓存数据是否存在
     *
     * @return {@link String}
     */
    @GetMapping("/exists")
    public String exists()
    {
        boolean exists = cacheChannel.exists(region, key);
        return "exists:" + exists;
    }

    /**
     * 清理指定区域的缓存
     *
     * @return {@link String}
     */
    @GetMapping("/clear")
    public String clear()
    {
        cache = null;
        cacheChannel.clear(region);
        return "clear success";
    }


    @Autowired
    private RedisUtils redisUtils;

    private Student queryMysqlById(long id)
    {
        log.info("查询Mysql数据库");
        try
        {
            Thread.sleep(10);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        Student student = new Student();
        student.setId(id);
        student.setName("张三");
        return student;
    }

    /**
     * 查询mysql ,测试缓存穿透
     *
     * @param id id
     * @return {@link Student}
```

```
     */
    private Student queryMysqlById2(long id)
    {
        log.info("查询Mysql数据库2");
        try
        {
            Thread.sleep(10);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        return null;
    }

    private Boolean updateMysqlById(Student student, long id)
    {
        log.info("更新Mysql数据库");
        try
        {
            Thread.sleep(10);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        return true;
    }




    @GetMapping("/query")
    public Student query()
    {
        Student result = redisUtils.query("tools:", "tools:lock:", 1L,
Student.class,
                this::queryMysqlById, 30L, TimeUnit.MINUTES, 60);
        return result;
    }

    @GetMapping("/query2")
    public Student query2()
    {
        Student result = redisUtils.query("tools:", "tools:lock:", 2L,
Student.class,
                this::queryMysqlById2, 30L, TimeUnit.MINUTES, 60);
        return result;
    }

    @GetMapping("/update")
    public boolean update()
    {
        Student student = new Student();
        student.setId(1L);
        student.setName("张三2");
        boolean update = redisUtils.update(1L, student, "tools:", s ->
updateMysqlById(student, 1L));
```

```
        return update;
    }



}

```

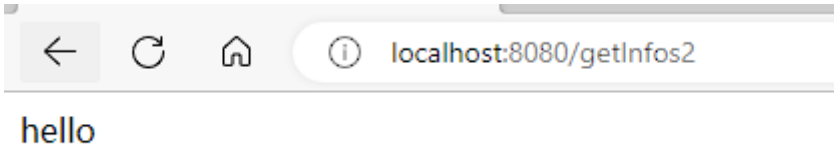
第五步：启动程序

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

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)
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
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"
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}"
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
2022-11-06 13:49:34.363  INFO 6148 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data Redis repositories in DEFAULT mode.
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.
2022-11-06 13:49:34.601  INFO 6148 --- [           main] mao.tools_j2cache.config.CacheConfig     : 初始化 CacheConfig
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)
```

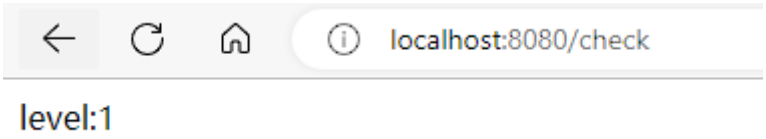
```
2022-11-06 13:49:34.845  INFO 6148 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-11-06 13:49:34.853  INFO 6148 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-11-06 13:49:34.853  INFO 6148 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-11-06 13:49:34.976  INFO 6148 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-11-06 13:49:34.976  INFO 6148 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1384 ms
2022-11-06 13:49:35.182  INFO 6148 --- [           main] n.o.j2cache.util.SerializationUtils      : Using Serializer -> [json:net.oschina.j2cache.util.FstJSONSerializer]
2022-11-06 13:49:35.185  INFO 6148 --- [           main] net.oschina.j2cache.CacheProviderHolder  : Using L1 CacheProvider : net.oschina.j2cache.caffeine.CaffeineProvider
2022-11-06 13:49:35.820  INFO 6148 --- [           main] net.oschina.j2cache.CacheProviderHolder  : Using L2 CacheProvider : net.oschina.j2cache.cache.support.redis.SpringRedisProvider
2022-11-06 13:49:35.832  INFO 6148 --- [           main] net.oschina.j2cache.J2CacheBuilder       : Using cluster policy : net.oschina.j2cache.cache.support.redis.SpringRedisPubSubPolicy
2022-11-06 13:49:35.854  INFO 6148 --- [           main] mao.tools_j2cache.config.RedissonConfig  : 初始化 RedissonConfig
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
2022-11-06 13:49:35.971  INFO 6148 --- [           main] org.redisson.Version                     : Redisson 3.17.0
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
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
2022-11-06 13:49:36.628  INFO 6148 --- [           main] m.tools_j2cache.config.RedisUtilsConfig  : 初始化 RedisUtilsConfig
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'
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 ''
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)
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'
2022-11-06 13:49:37.976  INFO 6148 --- [2)-172.18.144.1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
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

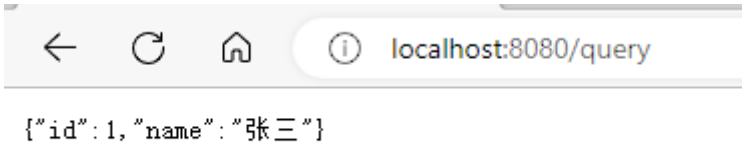


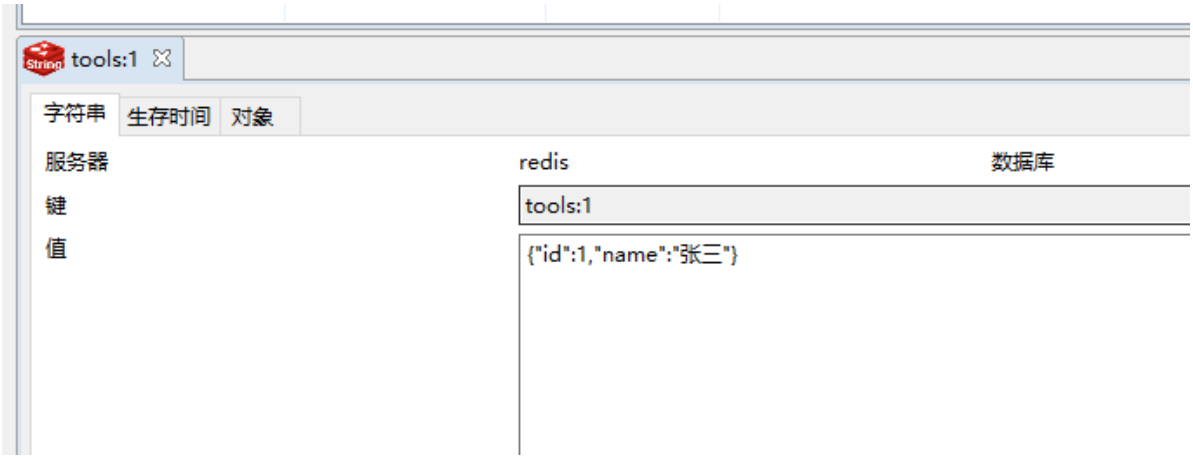
http://localhost:8080/check



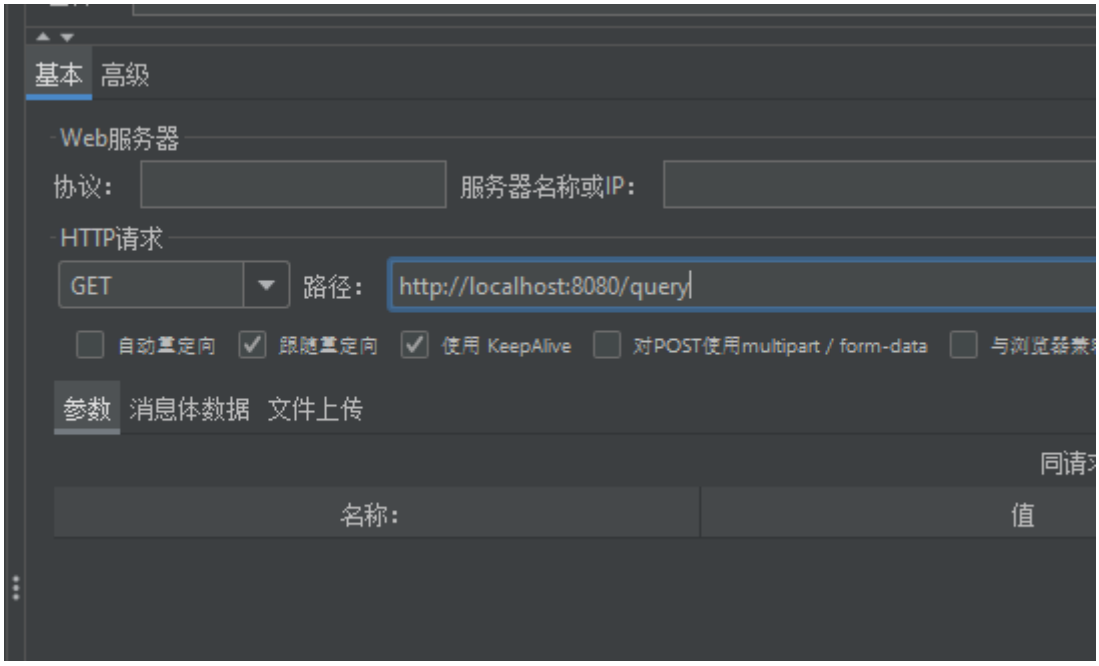
测试并发

http://localhost:8080/query



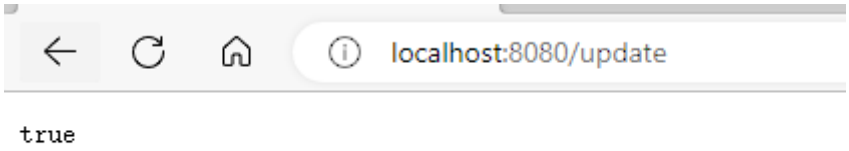

tools:1
字符串
生存时间
对象
服务器
redis
数据库
键
tools:1
值
{"id":1,"name":"张三"}


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
线程属性
线程数:
200
Ramp-Up时间（秒）:
1
循环次数
永远
Same user on each iteration
调度器
持续时间（秒）
启动延迟（秒）



尝试一次更新，让缓存过期

http://localhost:8080/update




```
- [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
```

```
2022-11-06 13:57:23.033  INFO 6148 --- [o-8080-exec-124] m.use_starter.controller.TestController  : 更新Mysql数据库
2022-11-06 13:57:23.062 DEBUG 6148 --- [o-8080-exec-124] mao.tools_j2cache.utils.RedisUtils       : 更新：tools:1
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
2022-11-06 13:57:23.197  INFO 6148 --- [io-8080-exec-26] m.use_starter.controller.TestController  : 查询Mysql数据库
```

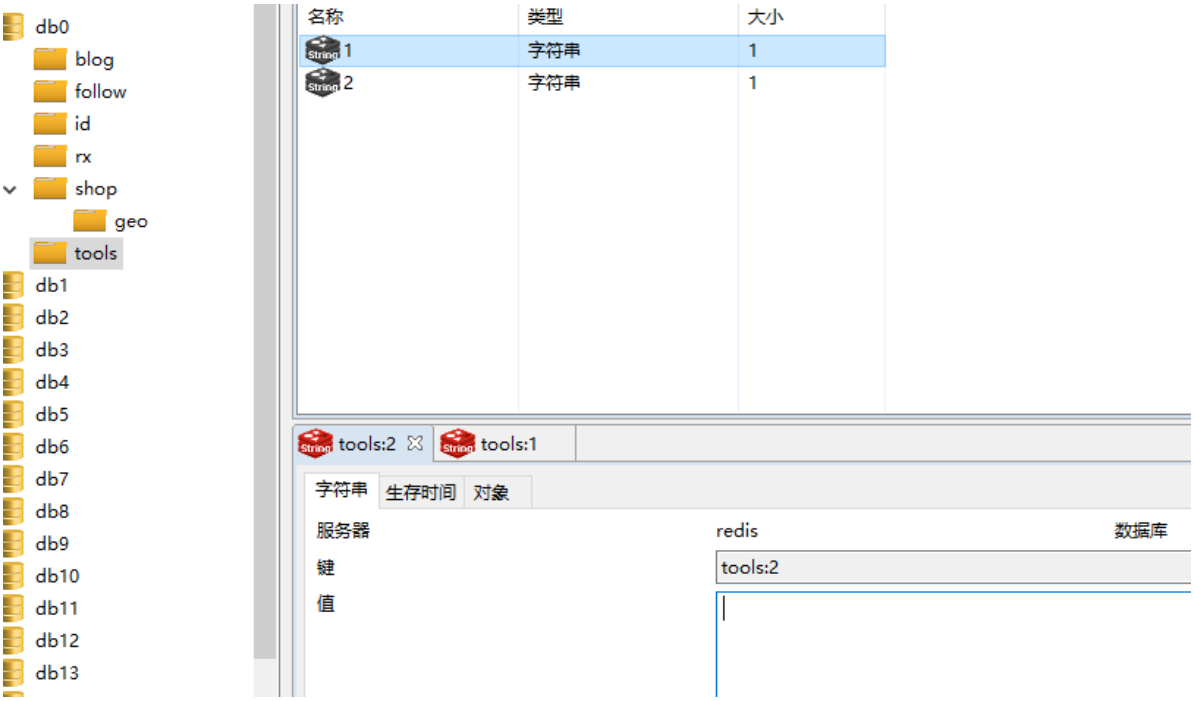
缓存过期后，只更新了一次数据库

测试缓存穿透

http://localhost:8080/query2

多次访问

数据库只查询了一次



缓存的是一个空字符串

```
C:\Users\mao>redis-cli
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> get tools:2
""
127.0.0.1:6379>
```

---

end

---

by mao
2022 11 06
