第五章节 Spring Boot 缓存技术

（SpringBoot 高级）

课程内容

- Spring Boot 整合 Ehcache
- Spring Boot 整合 Spring Data Redis

一、 Spring Boot 整合 Ehcache

1 修改 pom 文件

```
    <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
http://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>1.5.10.RELEASE</version>
        </parent>
        <groupId>com.bjsxt</groupId>
        <artifactId>23-spring-boot-ehcache</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <properties>
            <java.version>1.7</java.version>
            <thymeleaf.version>3.0.2.RELEASE</thymeleaf.version>

    <thymeleaf-layout-dialect.version>2.0.4</thymeleaf-layout-dialect.ve
rsion>
        </properties>

        <dependencies>
            <!-- springBoot 的启动器 -->
            <dependency>
```

```xml
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- springBoot 的启动器 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- springBoot 的启动器 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- 测试工具的启动器 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
    </dependency>

    <!-- mysql -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>

    <!-- druid 连接池 -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.0.9</version>
    </dependency>

    <!-- Spring Boot 缓存支持启动器 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-cache</artifactId>
    </dependency>

    <!-- Ehcache 坐标 -->
    <dependency>
        <groupId>net.sf.ehcache</groupId>
        <artifactId>ehcache</artifactId>
```

```
            </dependency>

        </dependencies>
    </project>
```

2 创建 Ehcache 的配置文件

文件名：ehcache.xml

位置：src/main/resources/ehcache.xml

```
    <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">

        <diskStore path="java.io.tmpdir"/>

      <!--defaultCache:echcache 的默认缓存策略  -->
        <defaultCache
                maxElementsInMemory="10000"
                eternal="false"
                timeToIdleSeconds="120"
                timeToLiveSeconds="120"
                maxElementsOnDisk="10000000"
                diskExpiryThreadIntervalSeconds="120"
                memoryStoreEvictionPolicy="LRU">
            <persistence strategy="localTempSwap"/>
        </defaultCache>
        <!-- 自定义缓存策略 -->
        <cache name="users"
                maxElementsInMemory="10000"
                eternal="false"
                timeToIdleSeconds="120"
                timeToLiveSeconds="120"
                maxElementsOnDisk="10000000"
                diskExpiryThreadIntervalSeconds="120"
                memoryStoreEvictionPolicy="LRU">
            <persistence strategy="localTempSwap"/>
        </cache>
    </ehcache>
```

3 修改 application.properties 文件

```
    spring.datasource.driverClassName=com.mysql.jdbc.Driver
    spring.datasource.url=jdbc:mysql://localhost:3306/ssm
    spring.datasource.username=root
```

```
spring.datasource.password=root

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.cache.ehcache.cofnig=ehcache.xml
```

4 修改启动类

```

@SpringBootApplication
@EnableCaching
public class App {

    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

5 创建业务层

```
/**
 * UsersService 接口实现类
 *
 *
 */
@Service
public class UsersServiceImpl implements UsersService {

    @Autowired
    private UsersRepository usersRepository;

    @Override
    public List<Users> findUserAll() {
        return this.usersRepository.findAll();
    }

    @Override
    //@Cacheable:对当前查询的对象做缓存处理
    @Cacheable(value="users")
    public Users findUserById(Integer id) {
        return this.usersRepository.findOne(id);
    }

    @Override
    public Page<Users> findUserByPage(Pageable pageable) {
        return this.usersRepository.findAll(pageable);
    }

    @Override
    public void saveUsers(Users users) {
        this.usersRepository.save(users);
    }

}
```

6 修改实体类 Users

```
@Entity
@Table(name="t_users")
public class Users implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private Integer id;

    @Column(name="name")
    private String name;

    @Column(name="age")
    private Integer age;

    @Column(name="address")
    private String address;


    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "Users [id=" + id + ", name=" + name + ", age=" + age +
", address=" + address + "]";
    }
}
```

7 测试

```
/**
 * UsersService 测试
 *
 *
 */
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes=App.class)
public class UsersServiceTest {

    @Autowired
```

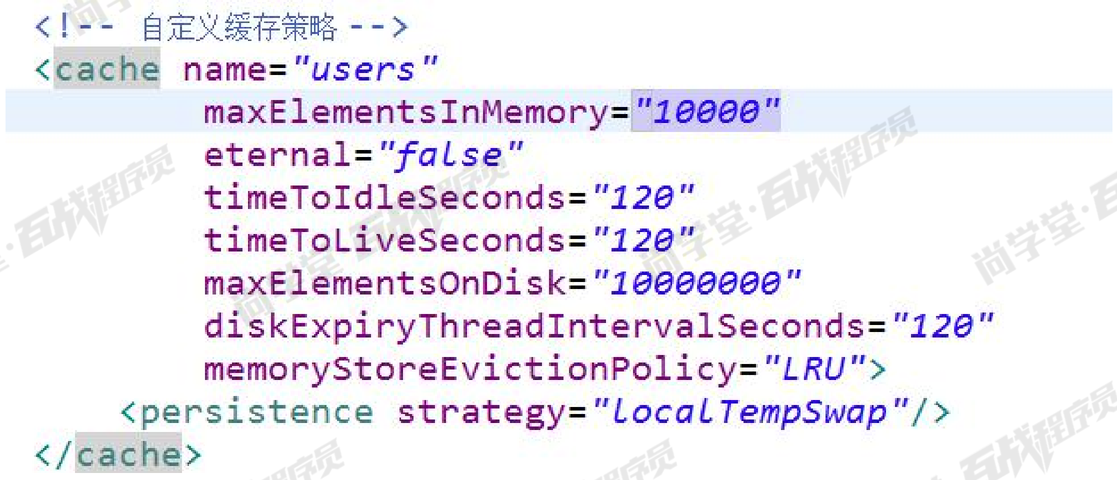
```java
    private UsersService usersService;

    @Test
    public void testFindUserById(){
        //第一次查询
        System.out.println(this.usersService.findUserById(1));

        //第二次查询
        System.out.println(this.usersService.findUserById(1));
    }
}
```

二、 @Cacheable 与@CacheEvict

1 @Cacheable

@Cacheable 作用：把方法的返回值添加到 Ehcache 中做缓存

Value 属性：指定一个 Ehcache 配置文件中的缓存策略，如果么有给定 value，name 则表示使用默认的缓存策略。



```xml
<!-- 自定义缓存策略 -->
<cache name="users"
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        maxElementsOnDisk="10000000"
        diskExpiryThreadIntervalSeconds="120"
        memoryStoreEvictionPolicy="LRU">
    <persistence strategy="localTempSwap"/>
</cache>
```

Key 属性：给存储的值起个名称。在查询时如果有名称相同的，那么则知已从缓存中将数据返回

1.1业务层

```java
@Override
    @Cacheable(value="users",key="#pageable.pageSize")
    public Page<Users> findUserByPage(Pageable pageable) {
        return this.usersRepository.findAll(pageable);
    }
```

1.2 测试代码

```
    @Test
        public void testFindUserByPage(){
            Pageable pageable = new PageRequest(0, 2);
            //第一次查询
            
    System.out.println(this.usersService.findUserByPage(pageable).getTot
alElements());
            
            //第二次查询
            
    System.out.println(this.usersService.findUserByPage(pageable).getTot
alElements());
            
            //第三次查询
            pageable = new PageRequest(1, 2);
            
    System.out.println(this.usersService.findUserByPage(pageable).getTot
alElements());
        }
    }
```

2 @CacheEvict

@CacheEvict 作用：清除缓存

2.1 业务层

```
    /**
     * UsersService 接口实现类
     *
     *
     */
    @Service
    public class UsersServiceImpl implements UsersService {
    
        @Autowired
        private UsersRepository usersRepository;
        
        @Override
        @Cacheable(value="users")
        public List<Users> findUserAll() {
            return this.usersRepository.findAll();
        }

        @Override
        //@Cacheable:对当前查询的对象做缓存处理
        @Cacheable(value="users")
        public Users findUserById(Integer id) {
            return this.usersRepository.findOne(id);
        }

        @Override
        @Cacheable(value="users",key="#pageable.pageSize")
        public Page<Users> findUserByPage(Pageable pageable) {
            return this.usersRepository.findAll(pageable);
        }

        @Override
        //@CacheEvict(value="users",allEntries=true) 清除缓存中以 users 缓
    存策略缓存的对象
        @CacheEvict(value="users",allEntries=true)
        public void saveUsers(Users users) {
            this.usersRepository.save(users);
        }

    }
```

2.2 测试代码

```
@Test
    public void testFindAll(){
        //第一次查询
        System.out.println(this.usersService.findUserAll().size());

        Users users = new Users();
        users.setAddress("南京");
        users.setAge(43);
        users.setName("朱七");
        this.usersService.saveUsers(users);

        //第二次查询
        System.out.println(this.usersService.findUserAll().size());
    }
```

三、 Spring Boot 整合 Spring Data Redis

Redis 版本：3.0.0
运行环境：Linux

1 安装 Redis

1.1 安装 gcc

```
Yum install gcc-c++
```

1.2 解压 redis.3.0.0.tar.gz 压缩包

```
tar -zxvf redis-3.0.0.tar.gz
```

1.3 进入解压后的目录进行编译

```
cd redis-3.0.0
make
```

1.4 将 Redis 安装到指定目录

```
make PREFIX=/usr/local/redis install
```

1.5 启动 Redis

```
./redis-server
```

2 Spring Boot 整合 Spring Data Redis

Spring Data Redis 是属于 Spring Data 下的一个模块。作用就是简化对于 redis 的操做

2.1 修改 pom 文件添加 Spring Data Redis 的坐标

```
<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
```

```
http://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>1.5.10.RELEASE</version>
      </parent>
      <groupId>com.bjsxt</groupId>
      <artifactId>24-spring-boot-redis</artifactId>
      <version>0.0.1-SNAPSHOT</version>

      <properties>
      <java.version>1.7</java.version>
      <thymeleaf.version>3.0.2.RELEASE</thymeleaf.version>

<thymeleaf-layout-dialect.version>2.0.4</thymeleaf-layout-dialect.versi
on>
      </properties>

      <dependencies>
        <!-- springBoot 的启动器 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- thymeleaf 的启动器 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>

        <!-- Spring Data Redis 的启动器 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
      </dependencies>
    </project>
```

2.2 编写 Spring Data Redis 的配置类（重点）

```
/**
 * 完成对 Redis 的整合的一些配置
```

```
    *
    *
    */
   @Configuration
   public class RedisConfig {

       /**
        * 1.创建 JedisPoolConfig 对象。在该对象中完成一些链接池配置
        *
        */
       @Bean
       public JedisPoolConfig jedisPoolConfig(){
           JedisPoolConfig config = new JedisPoolConfig();
           //最大空闲数
           config.setMaxIdle(10);
           //最小空闲数
           config.setMinIdle(5);
           //最大链接数
           config.setMaxTotal(20);

           return config;
       }

       /**
        * 2.创建 JedisConnectionFactory：配置 redis 链接信息
        */
       @Bean
       public JedisConnectionFactory
jedisConnectionFactory(JedisPoolConfig config){
           JedisConnectionFactory factory = new JedisConnectionFactory();
           //关联链接池的配置对象
           factory.setPoolConfig(config);
           //配置链接 Redis 的信息
           //主机地址
           factory.setHostName("192.168.70.128");
           //端口
           factory.setPort(6379);

           return factory;
       }

       /**
        * 3.创建 RedisTemplate:用于执行 Redis 操作的方法
        */
```

```java
        @Bean
        public RedisTemplate<String,Object>
redisTemplate(JedisConnectionFactory factory){
            RedisTemplate<String, Object> template = new RedisTemplate<>();
            //关联
            template.setConnectionFactory(factory);

            //为 key 设置序列化器
            template.setKeySerializer(new StringRedisSerializer());
            //为 value 设置序列化器
            template.setValueSerializer(new StringRedisSerializer());

            return template;
        }
    }
```

2.3 编写测试代码，测试整合环境

2.3.1 修改 pom 文件添加测试启动器坐标

```xml
     <!-- Test 的启动器 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
```

2.3.2 编写测试类

```java
    /**
     * Spring Data Redis 测试
     *
     *
     */
    @RunWith(SpringJUnit4ClassRunner.class)
    @SpringBootTest(classes=App.class)
    public class RedisTest {

        @Autowired
        private RedisTemplate<String, Object> redisTemplate;

        /**
         * 添加一个字符串
         */
        @Test
        public void testSet(){
            this.redisTemplate.opsForValue().set("key", "北京尚学堂");
        }

        /**
         * 获取一个字符串
         */
        @Test
        public void testGet(){
            String value = 
(String)this.redisTemplate.opsForValue().get("key");
            System.out.println(value);
        }
    }
```

3　提取 redis 的配置信息

3.1在 src/main/resource/ 目录下新建一个配置文件:application.properties

```
    spring.redis.pool.max-idle=10
    spring.redis.pool.min-idle=5
    spring.redis.pool.max-total=20

    spring.redis.hostName=192.168.70.128
    spring.redis.port=6379
```

3.2 修改配置类

```
    /**
     * 完成对 Redis 的整合的一些配置
     *
     *
     */
    @Configuration
    public class RedisConfig {

        /**
         * 1.创建 JedisPoolConfig 对象。在该对象中完成一些链接池配置
         * @ConfigurationProperties:会将前缀相同的内容创建一个实体。
     */
    @Bean
    @ConfigurationProperties(prefix="spring.redis.pool")
    public JedisPoolConfig jedisPoolConfig(){
        JedisPoolConfig config = new JedisPoolConfig();
        /*//最大空闲数
        config.setMaxIdle(10);
        //最小空闲数
        config.setMinIdle(5);
        //最大链接数
        config.setMaxTotal(20);*/
        System.out.println("默认值: "+config.getMaxIdle());
        System.out.println("默认值: "+config.getMinIdle());
        System.out.println("默认值: "+config.getMaxTotal());
        return config;
    }

    /**
     * 2.创建 JedisConnectionFactory: 配置 redis 链接信息
     */
    @Bean
    @ConfigurationProperties(prefix="spring.redis")
    public JedisConnectionFactory
jedisConnectionFactory(JedisPoolConfig config){
        System.out.println("配置完毕: "+config.getMaxIdle());
        System.out.println("配置完毕: "+config.getMinIdle());
        System.out.println("配置完毕: "+config.getMaxTotal());

        JedisConnectionFactory factory = new JedisConnectionFactory();
        //关联链接池的配置对象
        factory.setPoolConfig(config);
        //配置链接 Redis 的信息
        //主机地址
        /*factory.setHostName("192.168.70.128");
        //端口
        factory.setPort(6379);*/
        return factory;
    }

    /**
     * 3.创建 RedisTemplate:用于执行 Redis 操作的方法
     */
    @Bean
    public RedisTemplate<String,Object>
```

```
redisTemplate(JedisConnectionFactory factory){
           RedisTemplate<String, Object> template = new RedisTemplate<>();
           //关联
           template.setConnectionFactory(factory);

           //为 key 设置序列化器
           template.setKeySerializer(new StringRedisSerializer());
           //为 value 设置序列化器
           template.setValueSerializer(new StringRedisSerializer());

           return template;
       }
   }
```

4 Spring Data Redis 操作实体对象

4.1.1 创建实体类

```
   public class Users implements Serializable {

       private Integer id;
       private String name;
       private Integer age;
       public Integer getId() {
           return id;
       }
       public void setId(Integer id) {
           this.id = id;
       }
       public String getName() {
           return name;
       }
       public void setName(String name) {
           this.name = name;
       }
       public Integer getAge() {
           return age;
       }
       public void setAge(Integer age) {
           this.age = age;
       }
```

```
        @Override
        public String toString() {
            return "Users [id=" + id + ", name=" + name + ", age=" + age +
"]";
        }

    }
```

4.1.2　测试代码

```
    /**
         * 添加 Users 对象
         */
        @Test
        public void testSetUesrs(){
            Users users = new Users();
            users.setAge(20);
            users.setName("张三丰");
            users.setId(1);
            //重新设置序列化器
            this.redisTemplate.setValueSerializer(new
JdkSerializationRedisSerializer());
            this.redisTemplate.opsForValue().set("users", users);
        }

        /**
         * 取 Users 对象
         */
        @Test
        public void testGetUsers(){
            //重新设置序列化器
            this.redisTemplate.setValueSerializer(new
JdkSerializationRedisSerializer());
            Users users =
(Users)this.redisTemplate.opsForValue().get("users");
            System.out.println(users);
        }
```

5 Spring Data Redis 以 JSON 格式存储实体对象

5.1测试代码

```
    /**
         * 基于 JSON 格式存 Users 对象
         */
        @Test
        public void testSetUsersUseJSON(){
            Users users = new Users();
            users.setAge(20);
            users.setName("李四丰");
            users.setId(1);
            this.redisTemplate.setValueSerializer(new 
Jackson2JsonRedisSerializer<>(Users.class));
            this.redisTemplate.opsForValue().set("users_json", users);
        }

        /**
         * 基于 JSON 格式取 Users 对象
         */
        @Test
        public void testGetUseJSON(){
            this.redisTemplate.setValueSerializer(new 
Jackson2JsonRedisSerializer<>(Users.class));
            Users users = 
(Users)this.redisTemplate.opsForValue().get("users_json");
            System.out.println(users);
        }
```