芋道源码 2022-04-03

# Redis 缓存

`yudao-spring-boot-starter-redis` 技术组件，使用 Redis 实现缓存的功能，它有 2 种使用方式：

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

## 1. 编程式缓存

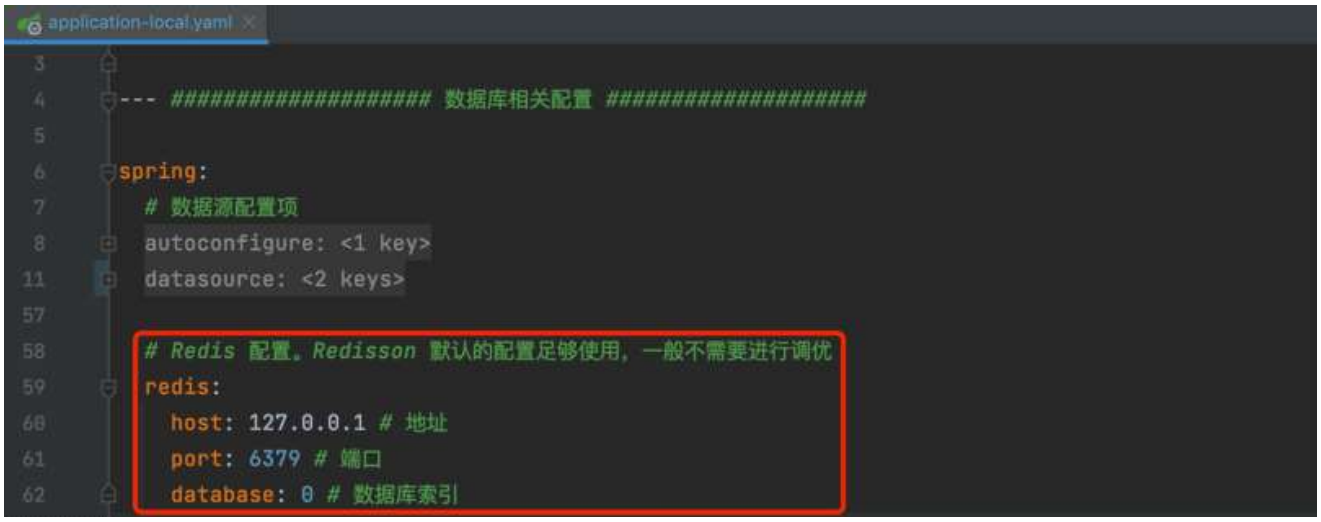
> **友情提示：**
>
> 如果你未学习过 Spring Data Redis 框架，可以后续阅读《芋道 Spring Boot Redis 入门》文章。

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

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

### 1.1 Spring Data Redis 配置

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



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

```java
@Configuration
public class YudaoRedisAutoConfiguration {

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

}
```

## 1.2 实战案例

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

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

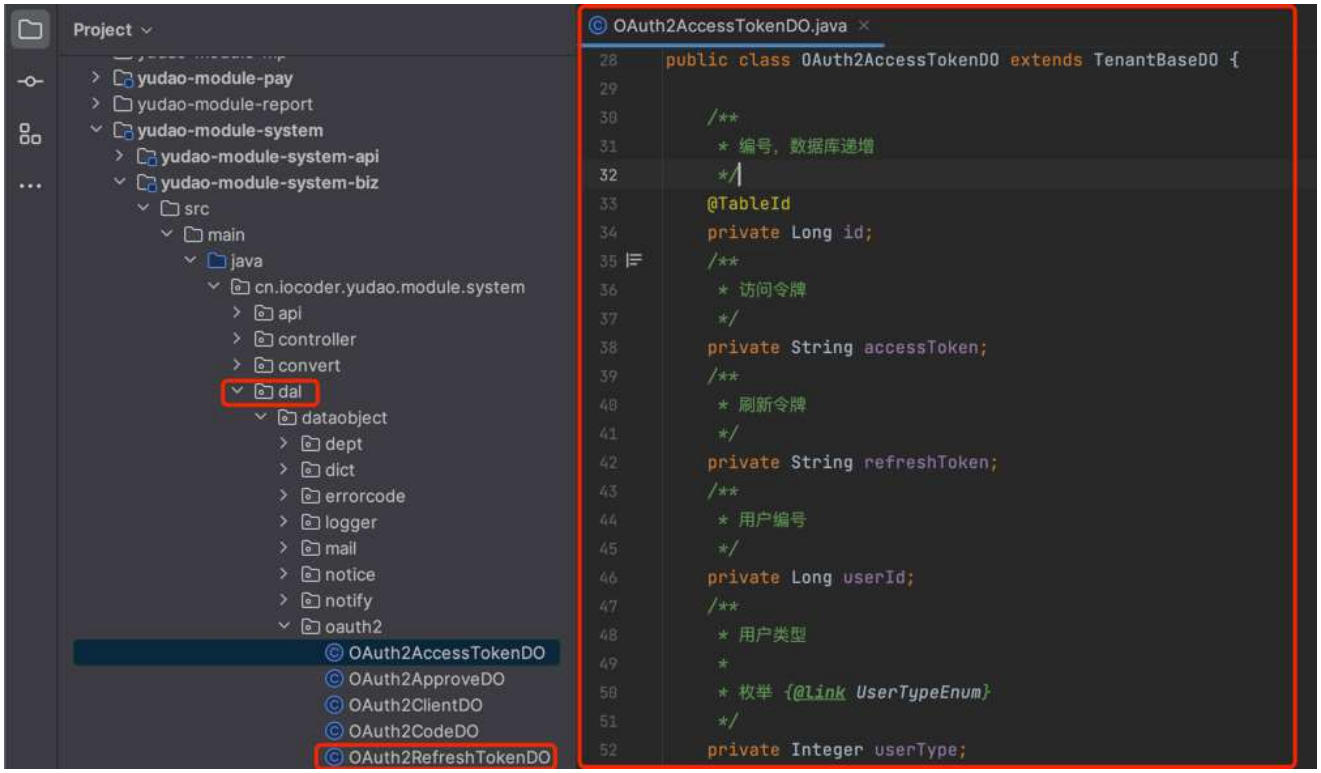
### 1.2.1 引入依赖

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

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

### 1.2.2 OAuth2AccessTokenDO

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

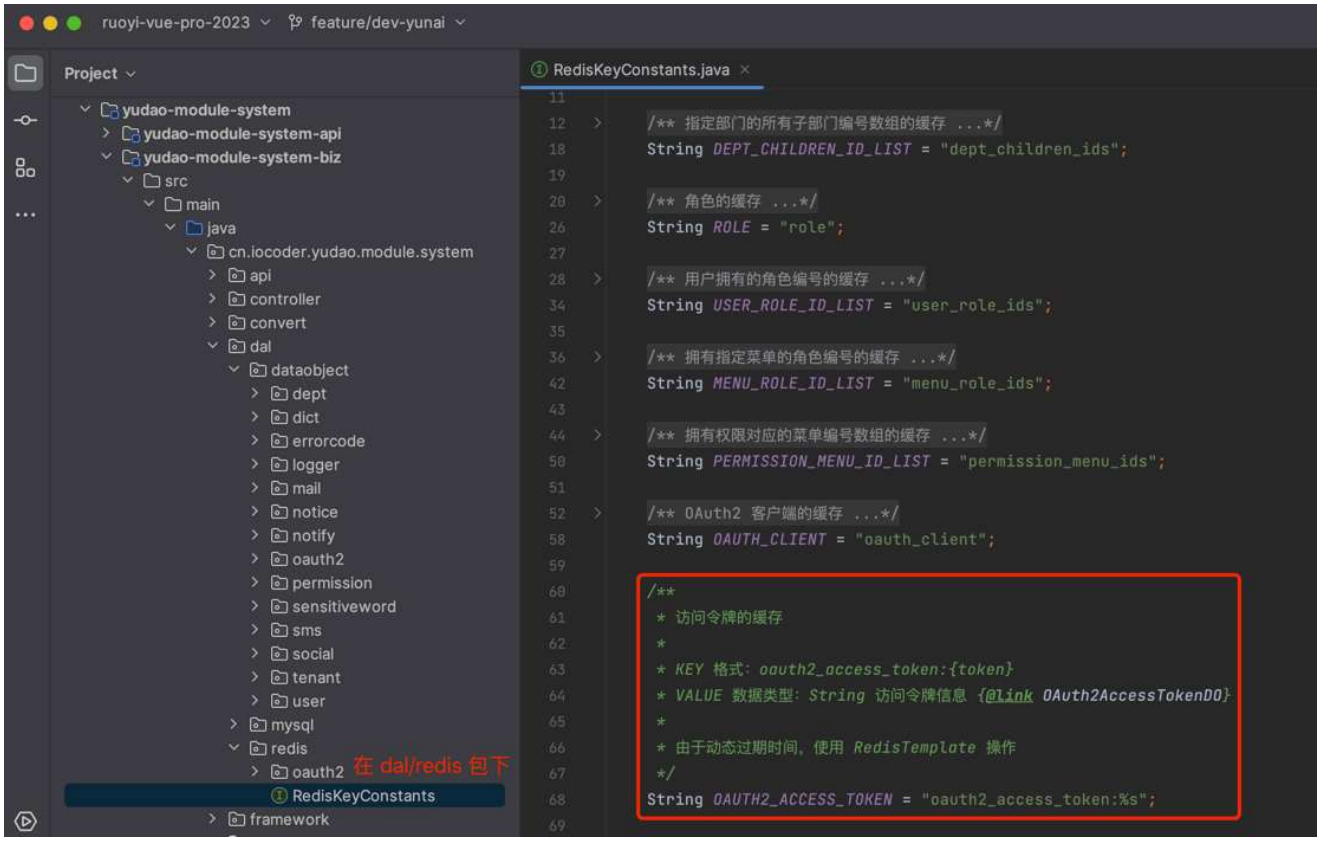



> **友情提示：**
>
> - ① 如果值是【简单】的 String 或者 Integer 等类型，无需创建数据实体。
> - ② 如果值是【复杂对象】时，建议在 `dal/dataobject` 包下，创建对应的数据实体。

### 1.2.3 RedisKeyConstants

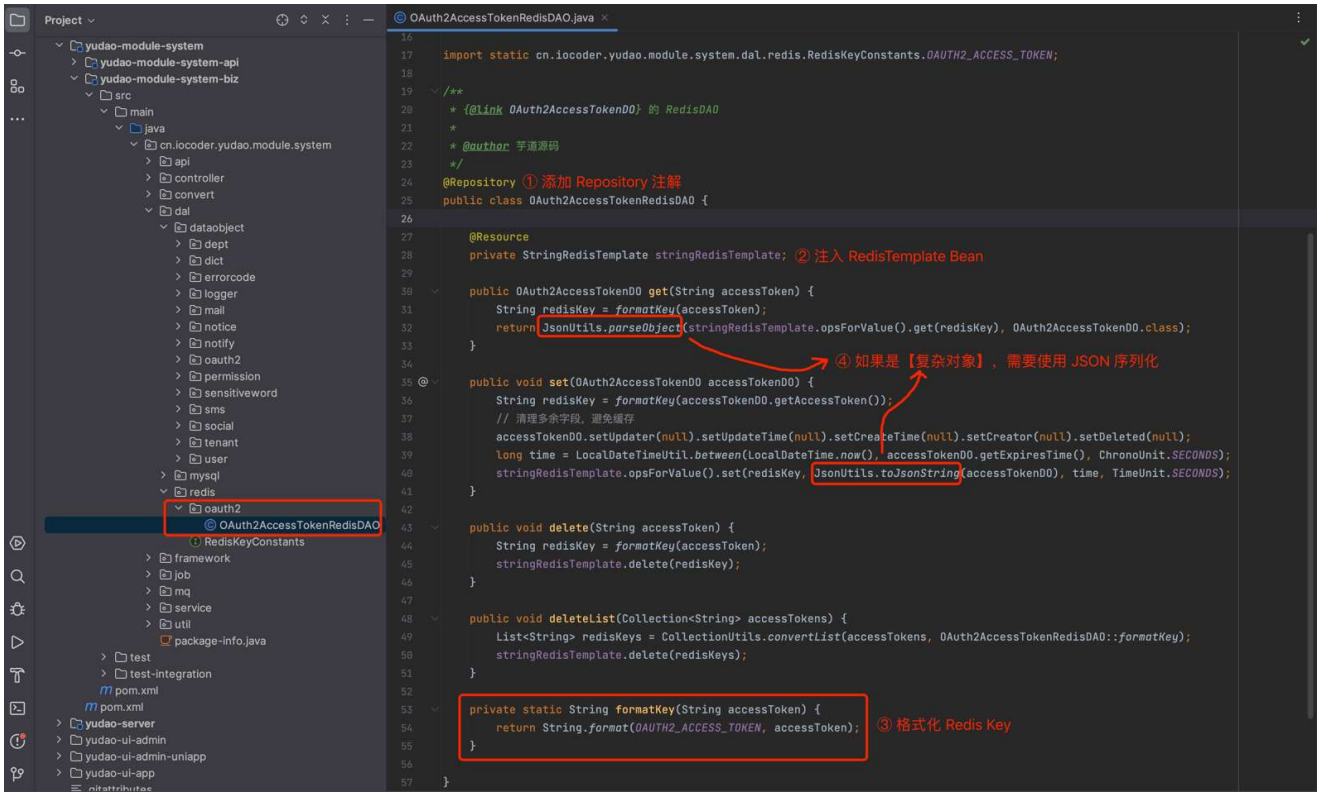
> **为什么要定义 Redis Key 常量？**
>
> 每个 `yudao-module-xxx` 模块，都有一个 RedisKeyConstants 类，定义该模块的 Redis Key 的信息。目的是，避免 Redis Key 散落在 Service 业务代码中，像对待数据库的表一样，对待每个 Redis Key。通过这样的方式，如果我们想要了解一个模块的 Redis 的使用情况，只需要查看 RedisKeyConstants 类即可。

在 `yudao-module-system` 模块的 RedisKeyConstants 类中，新建 OAuth2AccessTokenDO 对应的 Redis Key 定义 `OAUTH2_ACCESS_TOKEN` 。如下图所示：



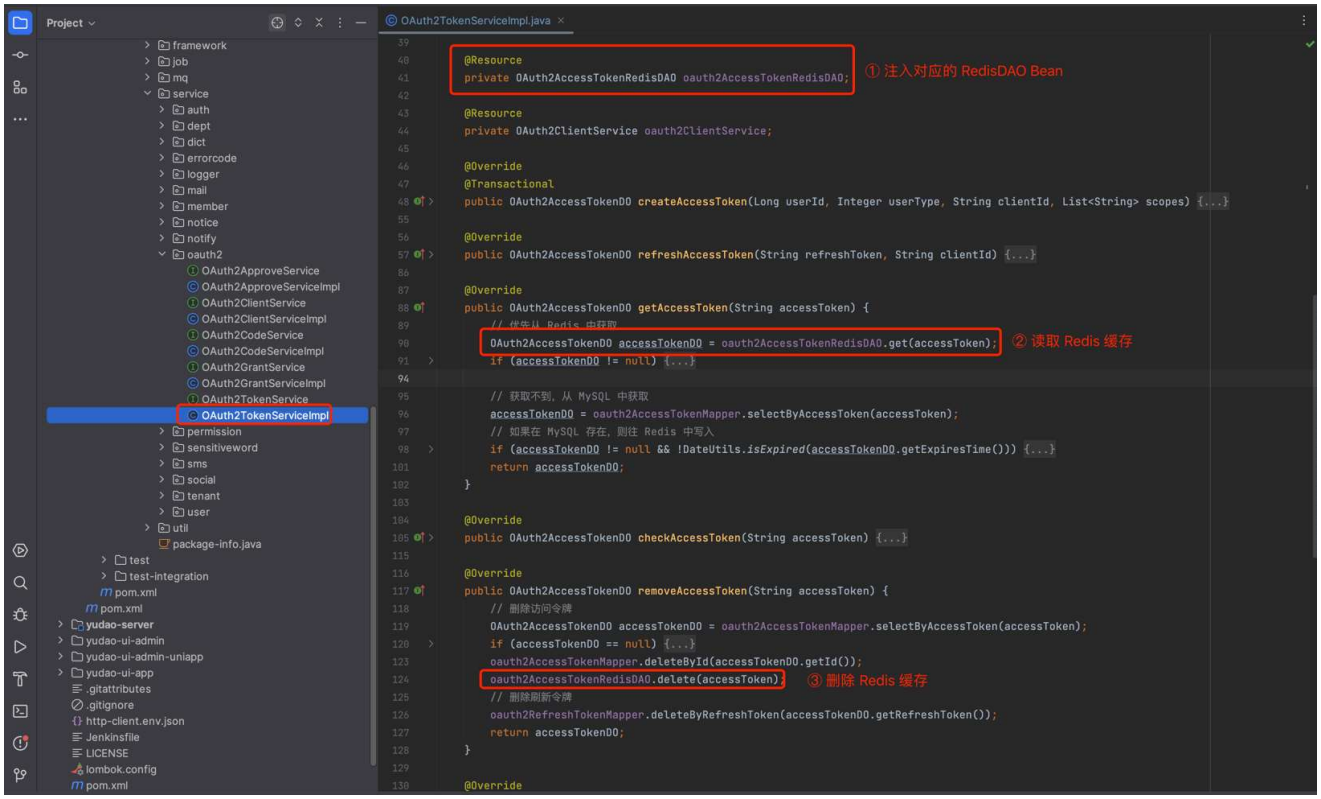

## 1.2.4 OAuth2AccessTokenRedisDAO

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




## 1.2.5 OAuth2TokenServiceImpl

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




## 2. 声明式缓存

> **友情提示：**
>
> 如果你未学习过 Spring Cache 框架，可以后续阅读《芋道 Spring Boot Cache 入门》文章。

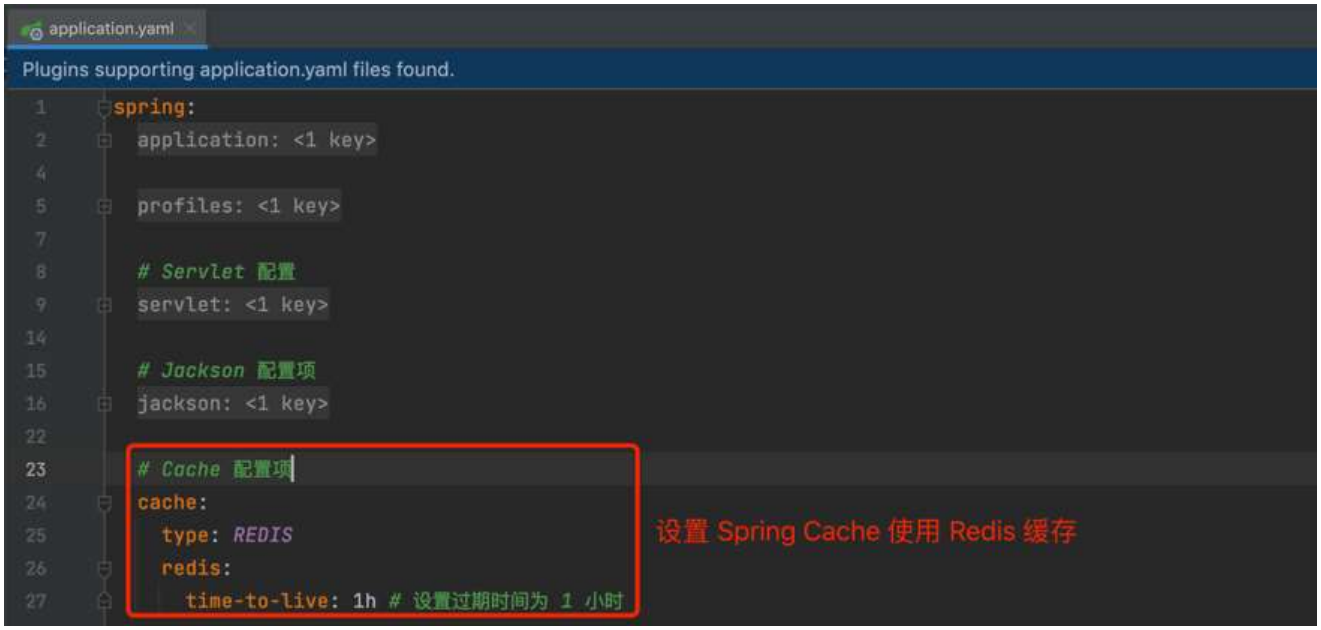
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

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

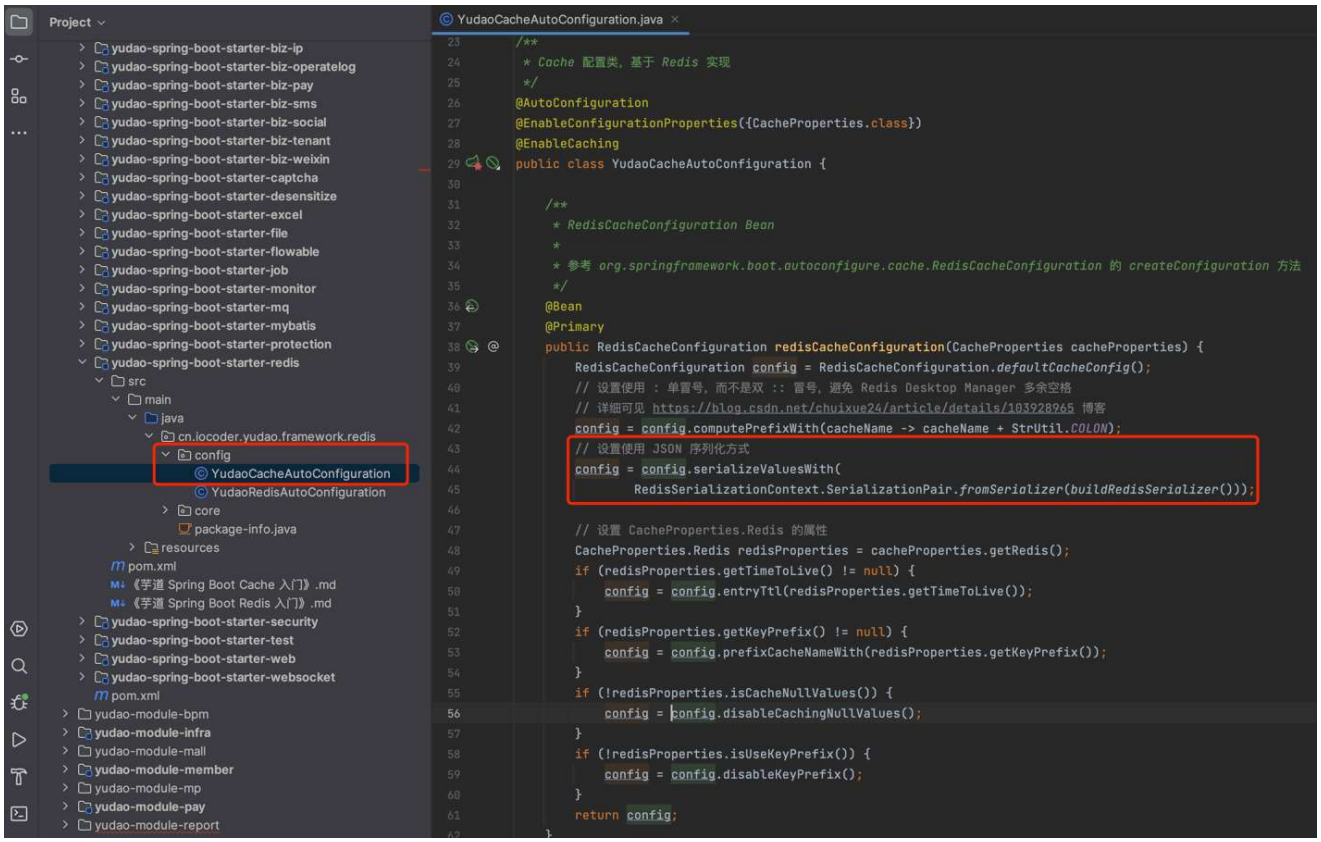
```
@Cacheable(value = "users", key = "#id")
UserDO getUserById(Integer id);
```

### 2.1 Spring Cache 配置

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




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




## 2.2 常见注解

### 2.2.1 @Cacheable 注解

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

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

### 2.2.2 @CachePut 注解

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

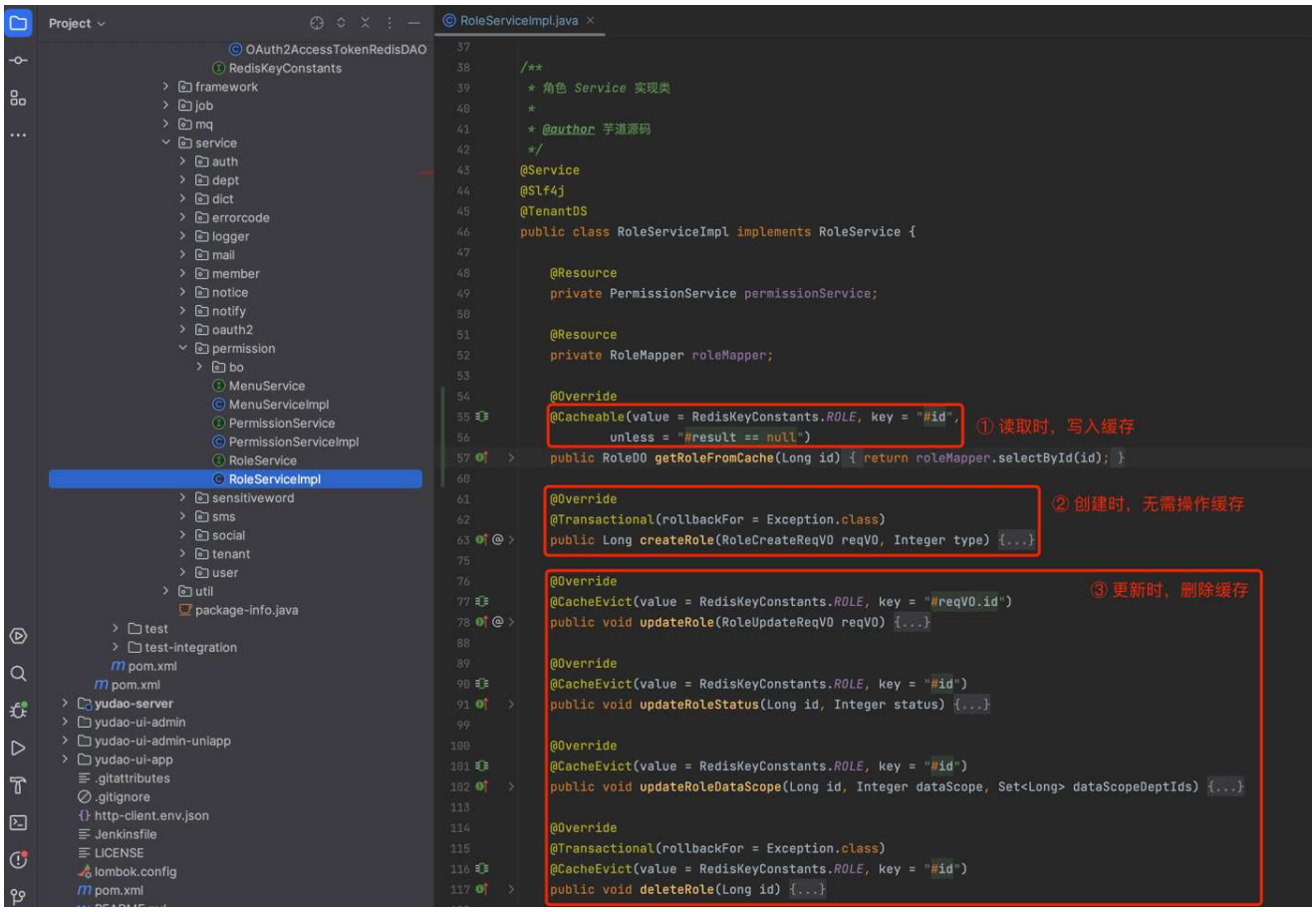
- 1）首先，执行方法，获得方法结果。也就是说，无论是否有缓存，都会执行方法。
- 2）然后，根据是否满足缓存的条件。如果满足，则缓存方法结果到缓存。
- 3）最后，返回方法结果。

### 2.2.3 @CacheEvict 注解

`@CacheEvict` 注解，添加在方法上，删除缓存。

## 2.3 实战案例

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




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

> **友情提示：**

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

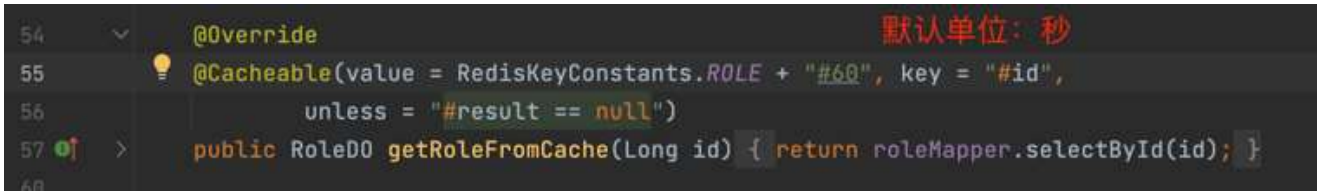
① 执行 `#getRoleFromCache(...)` 方法，从 MySQL 读取数据后，向 Redis 写入缓存。如下图所示：


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


## 2.4 过期时间

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

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

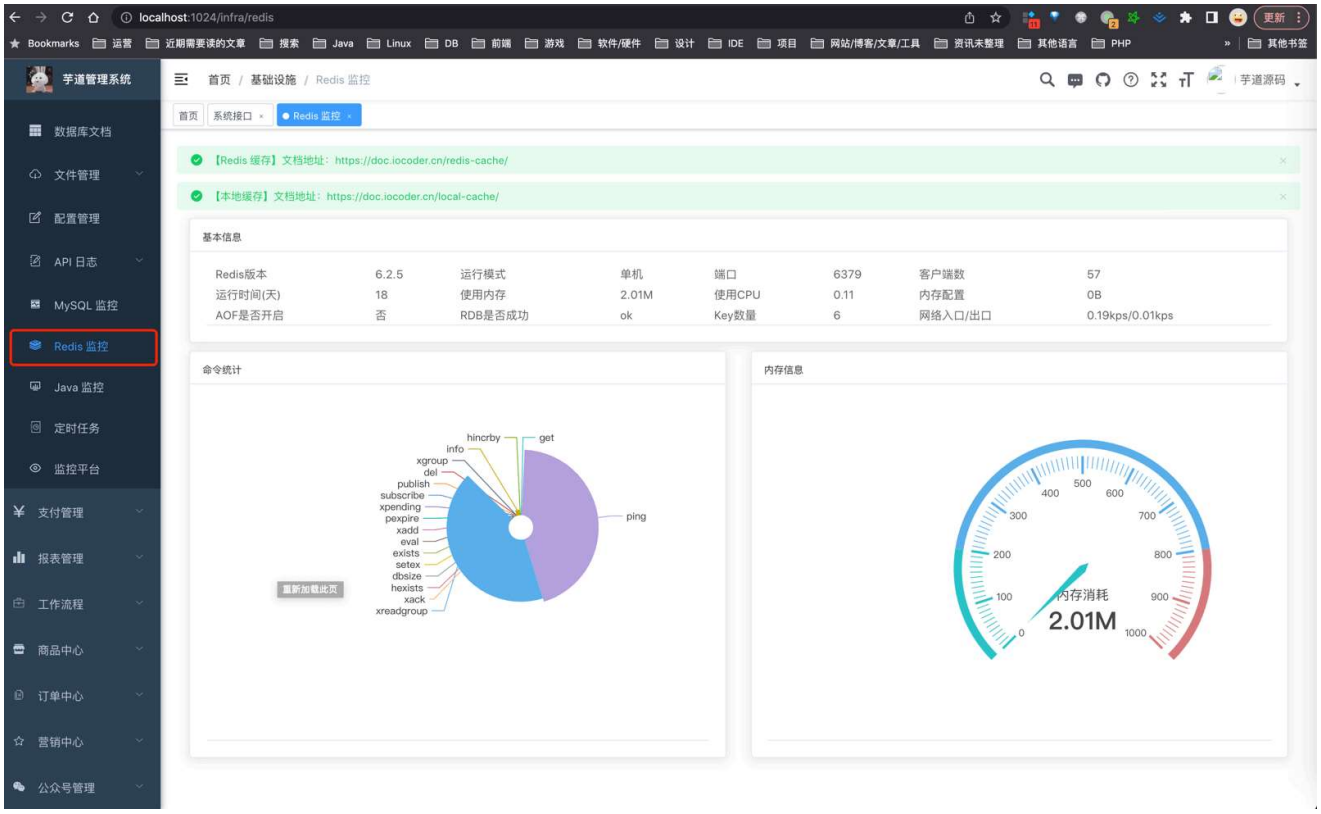



实现的原来，参考《Spring @Cacheable 扩展支持自定义过期时间》文章。

# 3. Redis 监控

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

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




← 多数据源（读写分离）、事务 本地缓存→