1. 导入依赖：

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

1. 在yml里面配置：

spring:

cache:  
 redis:  
 cache-null-values: true  
 key-prefix: ACTI\_  
 time-to-live: 3600000  
#设置空值存储，前缀，过期时间：毫秒为单位

1. 写配置文件

@EnableConfigurationProperties(CacheProperties.class)  
@Configuration  
@EnableCaching  
public class MyCacheConfig {  
 // @Autowired  
 // public CacheProperties cacheProperties;  
 /\*\*  
 \* 配置文件的配置没有用上  
 \* @return  
 \*/  
 @Bean  
 public RedisCacheConfiguration redisCacheConfiguration(CacheProperties cacheProperties) {  
  
 RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig();  
 // config = config.entryTtl();  
 config = config.serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(new StringRedisSerializer()));  
 config = config.serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(new GenericJackson2JsonRedisSerializer()));  
  
 CacheProperties.Redis redisProperties = cacheProperties.getRedis();  
 //将配置文件中所有的配置都生效  
 if (redisProperties.getTimeToLive() != null) {  
 config = config.entryTtl(redisProperties.getTimeToLive());  
 }  
 if (redisProperties.getKeyPrefix() != null) {  
 config = config.prefixKeysWith(redisProperties.getKeyPrefix());  
 }  
 if (!redisProperties.isCacheNullValues()) {  
 config = config.disableCachingNullValues();  
 }  
 if (!redisProperties.isUseKeyPrefix()) {  
 config = config.disableKeyPrefix();  
 }  
 return config;  
 }  
  
}

1. 直接注解使用，不需要启动类开启了：查询使用cacheable

@GetMapping("/member/list")  
@Cacheable(value = {"memberList"},key = "#root.methodName")  
public R getMemberList(HttpSession session){  
 System.out.println("查询了数据库");  
 Object member = session.getAttribute("Member");  
 System.out.println(member.toString());  
 return R.ok().put("memberList",memberFeignService.memberList()) ;  
}

1. 添加使用@CacheEvict

@GetMapping("/member/add/{userName}/{userPassword}")  
@CacheEvict(value = "memberList",key = "'getMemberList'")  
public R addMember(@PathVariable(value = "userName")String userName,  
 @PathVariable(value = "userPassword")String userPassword){  
 return memberFeignService.addMember(userName,userPassword);  
  
}

Spring-Cache缓存相关的问题：

1）读模式:三个问题都能解决

缓存穿透:查询一个null数据。解决:缓存空数据（yml配置有写）

缓存击穿:大量并发进来同时查询一个正好过期的数据。解决:加锁;（未知）

缓存雪崩:大量的key同时过期。解决:加过期时间。（yml上面也有写）

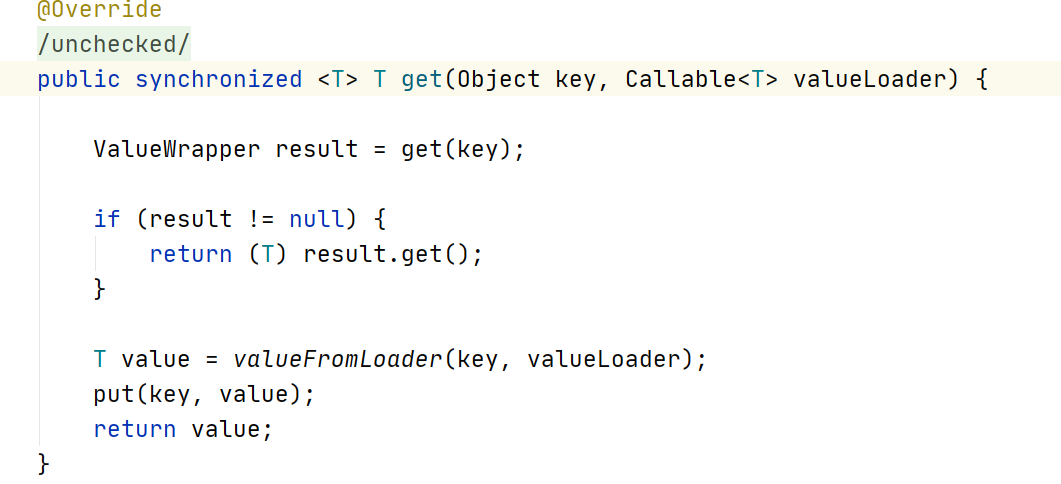
应对缓存击穿问题，查看RedisCacheManager，然后查看RedisCache发现get发现上面是有加sychorized锁的，他加的不是分布式锁，而是this锁，也叫synchronize锁，这样放请求只会放对应服务器数量（或者叫对应类）的请求进来。**但默认不开启加锁，但可以在注解上加上sync=true开启。**

@Cacheable(value = {"memberList"},key = "#root.methodName",sync = true)

CacheManager（RedisCacheManager）-》Cache（RedisCache）-》RedisCache负责缓存读写。

**没有开启锁的时候**：查询发起时，先走lookup方法，发现redis上没有缓存了就返回null，然后走构造器的doget方法，仍旧为空，继续返回null，就返回到业务逻辑方法，从业务逻辑里面得到数据后调用put方法，最后返回给用户值。

**开启锁的时候：**直接调用加锁的get请求，第一个get会使用未加锁的get方法，调取lookup查看缓存是否存在，抛出为空则使用valueFromLoader读取业务逻辑的数据，将其数据放到缓存里，然后返回这个数据给我们的用户



1. 写模式：spring-cache没有接管，只有过期时间。

1)、读写加锁。

2)、引入canal，感知到MysQL的更新去更新数据库

3)、读多写多,直接去数据库查询就行

总结

常规数据（读多写少，即时性，一致性要求不高的数据)﹔完全可以使用Spring-Cache。写模式：尽管spring-cache没有接管但只要有设置过期时间完全够用。

特殊数据:特殊设计