链滴

Spring Boot 1.5.x 整合 Redis

作者：HuixiaZhang
原文链接：https://ld246.com/article/1541668945116
来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180905.jpg?imageView2/1/w/960/h/520/interlace/1/q/100">

添加依赖
<pre><code class="highlight-chroma"><span class="highlight
cl"><dependency>
  <groupId>
rg.springframework.boot</groupId>
  <artifactId>
pring-boot-starter-data-redis</artifactId>
</dependency
gt;
</code></pre>
该依赖里默认包含了 spring-data-redis 和 Jedis 依赖，见<a href="https://ld246.com/forwar
?goto=https%3A%2F%2Fgithub.com%2Fspring-projects%2Fspring-boot%2Fblob%2F1.5.x%2
spring-boot-starters%2Fspring-boot-starter-data-redis%2Fpom.xml" target="_blank" rel="no
ollow ugc">这里

添加配置文件
<pre><code class="highlight-chroma"><span class="highlight
cl"># REDIS (RedisProperties)
Redis数据库索
（默认为0）
spring.redis.datab
se=0
# Redis服务器地址
spring.redis.host=
92.168.0.58
Redis服务器连
端口
spring.redis.port=
379
Redis服务器连
密码（默认为空）
spring.redis.pass
ord=
连接池最大连接
（使用负值表示没有限制）
spring.redis.pool.
ax-active=8
连接池最大阻塞
待时间（使用负值表示没有限制）
spring.redis.pool.
ax-wait=-1
连接池中的最大
闲连接
spring.redis.pool.
ax-idle=8
连接池中的最小
闲连接
spring.redis.pool.
in-idle=0

# 连接超时时间（秒）
spring.redis.timeout=0
</code></pre>

添加 cache 的配置类
spring boot 在 Spring Data Redis 提供了两个模板：

RedisTemplate
StringRedisTemplate

RedisTemplate 会使用 JdkSerializationRedisSerializer 处理数据，这意味着 key 和 value 都通过 Java 进行序列化。
StringRedisTemplate 默认会使用 StringRedisSerializer 处理数据。
要是操作字符串的话，用 StringRedisTemplate 就可以满足。但要是想要存储一个对象 Object，们就需要使用 RedisTemplate，并对 key 采用 String 序列化方式，对 value 采用 json 序列化方式这时候就需要对 redisTemplate 自定义配置
<pre><code class="highlight-chroma">/**
 * @author liyf
 * Created in 2018 11\8
**/
@Configuration
@EnableCaching/启用缓存，这个注解很重要；
public class RedisCacheConfig {

  /**
   * 功能描述: 缓存理器
   *
   * @param [redisTemplate]
   * @return org.springframework.cache.CacheManager
   * @author liyf
   **/
  @Bean
  public CacheManager cacheManager(RedisTemplate<?, ?> redisTemplate) {
    RedisCacheManager rcm = new RedisCacheManager(redisTemplate);
    // 设置缓存过时间（单位:秒）
    rcm.setDefaultExpiration(60 * 24);
    return rcm;
  }

  @Bean

```
</span></span><span class="highlight-line"><span class="highlight-cl">  public RedisTem
late&lt;String, String&gt; redisTemplate(RedisConnectionFactory factory) {
</span></span><span class="highlight-line"><span class="highlight-cl">    RedisTemplate
lt;String, String&gt; redisTemplate = new RedisTemplate&lt;&gt;();
</span></span><span class="highlight-line"><span class="highlight-cl">    redisTemplate.s
tConnectionFactory(factory);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    //key序列化方式
（不然会出现乱码;）,但是如果方法上有Long等非String类型的话，会报类型转换错误;
</span></span><span class="highlight-line"><span class="highlight-cl">    //所以在没有自
定义key生成策略的时候，以下这个代码建议不要这么写，可以不配置或者自己实现ObjectRedisSerial
zer
</span></span><span class="highlight-line"><span class="highlight-cl">    //或者JdkSerializ
tionRedisSerializer序列化方式;
</span></span><span class="highlight-line"><span class="highlight-cl">    RedisSerializer&
t;String&gt; redisSerializer = new StringRedisSerializer();//Long类型不可以会出现异常信息;
</span></span><span class="highlight-line"><span class="highlight-cl">    redisTemplate.s
tKeySerializer(redisSerializer);
</span></span><span class="highlight-line"><span class="highlight-cl">    redisTemplate.s
tHashKeySerializer(redisSerializer);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    return redisTe
plate;
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">  /**
</span></span><span class="highlight-line"><span class="highlight-cl">   * 功能描述: 自定
key.
</span></span><span class="highlight-line"><span class="highlight-cl">   *
</span></span><span class="highlight-line"><span class="highlight-cl">   * @param []
</span></span><span class="highlight-line"><span class="highlight-cl">    * @return org.s
ringframework.cache.interceptor.KeyGenerator
</span></span><span class="highlight-line"><span class="highlight-cl">   * @author liyf
</span></span><span class="highlight-line"><span class="highlight-cl">   */
</span></span><span class="highlight-line"><span class="highlight-cl">  @Override
</span></span><span class="highlight-line"><span class="highlight-cl">  public KeyGener
tor keyGenerator() {
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n("RedisCacheConfig.keyGenerator()");
</span></span><span class="highlight-line"><span class="highlight-cl">    return (o, meth
d, objects) -&gt; {
</span></span><span class="highlight-line"><span class="highlight-cl">    // This will gene
ate a unique key of the class name, the method name
</span></span><span class="highlight-line"><span class="highlight-cl">    //and all metho
 parameters appended.  StringBuilder sb = new StringBuilder();
</span></span><span class="highlight-line"><span class="highlight-cl">    String[] value =
new String[1];
</span></span><span class="highlight-line"><span class="highlight-cl">    // sb.append(ta
get.getClass().getName());
</span></span><span class="highlight-line"><span class="highlight-cl">    // sb.append(":"
+ method.getName());  Cacheable cacheable = method.getAnnotation(Cacheable.class);
</span></span><span class="highlight-line"><span class="highlight-cl">    if (cacheable !=
null) {
</span></span><span class="highlight-line"><span class="highlight-cl">    value = cacheab
```

```
e.value();
    }
    CachePut cachePut = method.getAnnotation(CachePut.class);
    if (cachePut != null) {
        value = cachePut.value();
    }
    CacheEvict cacheEvict = method.getAnnotation(CacheEvict.class);
    if (cacheEvict != null) {
        value = cacheEvict.value();
    }
    sb.append(value[0]);
    for (Object obj : objects) {
        sb.append(obj.toString());
    }

    return sb.toString();
    };
  }
}
```

注意:
要缓存的 Java 对象必须实现 Serializable 接口，因为 Spring 会将对象先序列化再存入 Redis，如果实现 Serializable 的话将会遇到类似这种错误：`nested exception is java.lang.IllegalArgumentException: DefaultSerializer requires a Serializable payload but received an object of type xx`

通过注解对数据进行缓存处理

```
@Service
public class UserServiceImpl implements UserService {

  private final UseRepository userRepository;

  @Autowired
  public UserServieImpl(UserRepository userRepository) {
    this.userReposiory = userRepository;
  }
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">  @CachePut(value = "userInfo", key = "#root.caches[0].name + #user.uid")
</span></span><span class="highlight-line"><span class="highlight-cl">  @Override
</span></span><span class="highlight-line"><span class="highlight-cl">  public User saveUser(User user) {
</span></span><span class="highlight-line"><span class="highlight-cl">    return userRepository.save(user);
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">  @Override
</span></span><span class="highlight-line"><span class="highlight-cl">  @Cacheable(value = "userInfo")
</span></span><span class="highlight-line"><span class="highlight-cl">  public User getUser(Long uid) {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.println("从数据库中进行获取的....uid=" + uid);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    return userRepository.findOne(uid);
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
</li>
<li>
<p>测试</p>
</li>
</ol>
```