<!-- spring-redis实现 -->

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

<version>1.6.2.RELEASE</version>

</dependency>

<!-- redis客户端jar -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.8.0</version>

</dependency>

<!-- Ehcache实现,用于参考 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-ehcache</artifactId>

<version>1.0.0</version>

</dependency>

<!-- 引入数据库配置文件 -->

<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="locations">

<list>

<value>classpath:jdbc.properties</value>

<value>classpath:redis.properties</value>

</list>

</property>

</bean>

<!-- redis数据源 -->

<bean id="poolConfig" class="redis.clients.jedis.JedisPoolConfig">

<property name="maxIdle" value="${redis.maxIdle}" />

<property name="maxTotal" value="${redis.maxActive}" />

<property name="maxWaitMillis" value="${redis.maxWait}" />

<property name="testOnBorrow" value="${redis.testOnBorrow}" />

</bean>

<!-- Spring-redis连接池管理工厂 -->

<bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"

p:host-name="${redis.host}" p:port="${redis.port}" p:password="${redis.pass}" p:pool-config-ref="poolConfig"/>

<!-- 使用中间类解决RedisCache.jedisConnectionFactory的静态注入，从而使MyBatis实现第三方缓存 -->

<bean id="redisCacheTransfer" class="com.strive.cms.cache.RedisCacheTransfer">

<property name="jedisConnectionFactory" ref="jedisConnectionFactory"/>

</bean>

Mybatis全局配置

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"<http://mybatis.org/dtd/mybatis-3-config.dtd>">

<configuration>

<!-- 配置mybatis的缓存，延迟加载等等一系列属性 -->

<settings>

<!-- 全局映射器启用缓存 -->

<setting name="cacheEnabled" value="true"/>

<!-- 查询时，关闭关联对象即时加载以提高性能 -->

<setting name="lazyLoadingEnabled" value="false"/>

<!-- 对于未知的SQL查询，允许返回不同的结果集以达到通用的效果 -->

<setting name="multipleResultSetsEnabled" value="true"/>

<!-- 允许使用列标签代替列名 -->

<setting name="useColumnLabel" value="true"/>

<!-- 不允许使用自定义的主键值(比如由程序生成的UUID 32位编码作为键值)，数据表的PK生成策略将被覆盖 -->

<setting name="useGeneratedKeys" value="false"/>

<!-- 给予被嵌套的resultMap以字段-属性的映射支持 FULL,PARTIAL -->

<setting name="autoMappingBehavior" value="PARTIAL"/>

<!-- 对于批量更新操作缓存SQL以提高性能 BATCH,SIMPLE -->

<!-- <setting name="defaultExecutorType" value="BATCH" /> -->

<!-- 数据库超过25000秒仍未响应则超时 -->

<!-- <setting name="defaultStatementTimeout" value="25000" /> -->

<!-- Allows using RowBounds on nested statements -->

<setting name="safeRowBoundsEnabled" value="false"/>

<!-- Enables automatic mapping from classic database column names A\_COLUMN to camel case classic Java property names aColumn. -->

<setting name="mapUnderscoreToCamelCase" value="true"/>

<!-- MyBatis uses local cache to prevent circular references and speed up repeated nested queries. By default (SESSION) all queries executed during a session are cached. If localCacheScope=STATEMENT

local session will be used just for statement execution, no data will be shared between two different calls to the same SqlSession. -->

<setting name="localCacheScope" value="SESSION"/>

<!-- Specifies the JDBC type for null values when no specific JDBC type was provided for the parameter. Some drivers require specifying the column JDBC type but others work with generic values

like NULL, VARCHAR or OTHER. -->

<setting name="jdbcTypeForNull" value="OTHER"/>

<!-- Specifies which Object's methods trigger a lazy load -->

<setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>

<!-- 设置关联对象加载的形态，此处为按需加载字段(加载字段由SQL指 定)，不会加载关联表的所有字段，以提高性能 -->

<setting name="aggressiveLazyLoading" value="true"/>

</settings>

</configuration> <!-- Allows using RowBounds on nested statements -->

<setting name="safeRowBoundsEnabled" value="false"/>

<!-- Enables automatic mapping from classic database column names A\_COLUMN to camel case classic Java property names aColumn. -->

<setting name="mapUnderscoreToCamelCase" value="true"/>

<!-- MyBatis uses local cache to prevent circular references and speed up repeated nested queries. By default (SESSION) all queries executed during a session are cached. If localCacheScope=STATEMENT

local session will be used just for statement execution, no data will be shared between two different calls to the same SqlSession. -->

<setting name="localCacheScope" value="SESSION"/>

<!-- Specifies the JDBC type for null values when no specific JDBC type was provided for the parameter. Some drivers require specifying the column JDBC type but others work with generic values

like NULL, VARCHAR or OTHER. -->

<setting name="jdbcTypeForNull" value="OTHER"/>

<!-- Specifies which Object's methods trigger a lazy load -->

<setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>

<!-- 设置关联对象加载的形态，此处为按需加载字段(加载字段由SQL指 定)，不会加载关联表的所有字段，以提高性能 -->

<setting name="aggressiveLazyLoading" value="true"/>

</settings>

</configuration>

**package** cn.cwx521.coom;  
  
**import** java.util.concurrent.locks.ReadWriteLock;  
**import** java.util.concurrent.locks.ReentrantReadWriteLock;  
  
**import** org.apache.ibatis.cache.Cache;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.data.redis.connection.jedis.JedisConnection;  
**import** org.springframework.data.redis.connection.jedis.JedisConnectionFactory;  
**import** org.springframework.data.redis.serializer.JdkSerializationRedisSerializer;  
**import** org.springframework.data.redis.serializer.RedisSerializer;  
  
**import** redis.clients.jedis.exceptions.JedisConnectionException;  
  
*/\*\*  
 \** ***@描述:*** *使用第三方内存数据库Redis作为二级缓存  
 \** ***@版权:*** *Copyright (c) 2016  
 \** ***@作者:*** *xiad  
 \** ***@版本:*** *1.0  
 \** ***@创建日期:*** *2016年3月2日  
 \** ***@创建时间:*** *下午8:02:57  
 \*/***public class** RedisCache **implements** Cache {  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(RedisCache.**class**);  
  
 **private static** JedisConnectionFactory *jedisConnectionFactory*;  
  
 **private final** String **id**;  
  
 */\*\*  
 \* The {****@code*** *ReadWriteLock}.  
 \*/* **private final** ReadWriteLock **readWriteLock** = **new** ReentrantReadWriteLock();  
  
 **public** RedisCache(**final** String id) {  
 **if** (id == **null**) {  
 **throw new** IllegalArgumentException(**"Cache instances require an ID"**);  
 }  
 ***logger***.debug(**"MybatisRedisCache:id="** + id);  
 **this**.**id** = id;  
 }  
  
 @Override  
 **public void** clear() {  
 JedisConnection connection = **null**;  
 **try** {  
 connection = *jedisConnectionFactory*.getConnection();  
 connection.flushDb();  
 connection.flushAll();  
 } **catch** (JedisConnectionException e) {  
 e.printStackTrace();  
 } **finally** {  
 **if** (connection != **null**) {  
 connection.close();  
 }  
 }  
 }  
  
 @Override  
 **public** String getId() {  
 **return this**.**id**;  
 }  
  
 @Override  
 **public** Object getObject(Object key) {  
 Object result = **null**;  
 JedisConnection connection = **null**;  
 **try** {  
 connection = *jedisConnectionFactory*.getConnection();  
 RedisSerializer<Object> serializer = **new** JdkSerializationRedisSerializer();  
 result = serializer.deserialize(connection.get(serializer.serialize(key)));  
 } **catch** (JedisConnectionException e) {  
 e.printStackTrace();  
 } **finally** {  
 **if** (connection != **null**) {  
 connection.close();  
 }  
 }  
 **return** result;  
 }  
 @Override  
 **public** ReadWriteLock getReadWriteLock() {  
 **return this**.**readWriteLock**;  
 }  
 @Override  
 **public int** getSize() {  
 **int** result = 0;  
 JedisConnection connection = **null**;  
 **try** {  
 connection = *jedisConnectionFactory*.getConnection();  
 result = Integer.*valueOf*(connection.dbSize().toString());  
 } **catch** (JedisConnectionException e) {  
 e.printStackTrace();  
 } **finally** {  
 **if** (connection != **null**) {  
 connection.close();  
 }  
 }  
 **return** result;  
 }  
  
 @Override  
 **public void** putObject(Object key, Object value) {  
 JedisConnection connection = **null**;  
 **try** {  
 connection = *jedisConnectionFactory*.getConnection();  
 RedisSerializer<Object> serializer = **new** JdkSerializationRedisSerializer();  
 connection.set(serializer.serialize(key), serializer.serialize(value));  
 } **catch** (JedisConnectionException e) {  
 e.printStackTrace();  
 } **finally** {  
 **if** (connection != **null**) {  
 connection.close();  
 }  
 }  
 }  
  
 @Override  
 **public** Object removeObject(Object key) {  
 JedisConnection connection = **null**;  
 Object result = **null**;  
 **try** {  
 connection = *jedisConnectionFactory*.getConnection();  
 RedisSerializer<Object> serializer = **new** JdkSerializationRedisSerializer();  
 result = connection.expire(serializer.serialize(key), 0);  
 } **catch** (JedisConnectionException e) {  
 e.printStackTrace();  
 } **finally** {  
 **if** (connection != **null**) {  
 connection.close();  
 }  
 }  
 **return** result;  
 }  
  
 **public static void** setJedisConnectionFactory(JedisConnectionFactory jedisConnectionFactory) {  
 RedisCache.*jedisConnectionFactory* = jedisConnectionFactory;  
 }  
}

**package** cn.cwx521.coom;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.data.redis.connection.jedis.JedisConnectionFactory;  
**public class** RedisCacheTransfer {  
 @Autowired  
 **public void** setJedisConnectionFactory(JedisConnectionFactory jedisConnectionFactory) {  
 RedisCache.*setJedisConnectionFactory*(jedisConnectionFactory);  
}}

properties

**jedis.host**=**localhost  
jedis.port**=**6379  
jedis.pass**=  
*#<!-- 最大连接数 -->***jedis.maxTotal**=**30***#最大空闲连接数***jedis.maxIdle**=**10***#<!-- 每次释放连接的最大数目 -->***jedis.numTestsPerEvictionRun**=**1024***#释放连接的扫描间隔（毫秒）***jedis.timeBetweenEvictionRunsMillis**=**3000***#连接最小空闲时间***jedis.minEvictableIdleTimeMillis**=**180000***#连接空闲多久后释放, 当空闲时间>该值 且 空闲连接>最大空闲连接数 时直接释放***jedis.softMinEvictableIdleTimeMillis**=**10000***#获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1***jedis.maxWaitMillis**=**1500***#在获取连接的时候检查有效性, 默认false***jedis.testOnBorrow**=**true***#在空闲时检查有效性, 默认false***jedis.testWhileIdle**=**true***#连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true***jedis.blockWhenExhausted**=**false**

applicataion-redis.xml

<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:p="http://www.springframework.org/schema/p" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.2.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context-3.2.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop-3.2.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx-3.2.xsd "**>  
 *<!--读取数据库配置-->* <**context:property-placeholder location="classpath:properties/\*.properties"**></**context:property-placeholder**>  
  
 <**bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig"**>  
 <**property name="maxTotal" value="${jedis.maxTotal}"**/>  
 *<!-- 最大空闲连接数 -->* <**property name="maxIdle" value="${jedis.maxIdle}"**/>  
 *<!-- 每次释放连接的最大数目 -->* <**property name="numTestsPerEvictionRun" value="${jedis.numTestsPerEvictionRun}"**/>  
 *<!-- 释放连接的扫描间隔（毫秒） -->* <**property name="timeBetweenEvictionRunsMillis" value="${jedis.timeBetweenEvictionRunsMillis}"**/>  
 *<!-- 连接最小空闲时间 -->* <**property name="minEvictableIdleTimeMillis" value="${jedis.minEvictableIdleTimeMillis}"**/>  
 *<!-- 连接空闲多久后释放, 当空闲时间>该值 且 空闲连接>最大空闲连接数 时直接释放 -->* <**property name="softMinEvictableIdleTimeMillis" value="${jedis.softMinEvictableIdleTimeMillis}"**/>  
 *<!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -->* <**property name="maxWaitMillis" value="${jedis.maxWaitMillis}"**/>  
 *<!-- 在获取连接的时候检查有效性, 默认false -->* <**property name="testOnBorrow" value="${jedis.testOnBorrow}"**/>  
 *<!-- 在空闲时检查有效性, 默认false -->* <**property name="testWhileIdle" value="${jedis.testWhileIdle}"**/>  
 *<!-- 连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true -->* <**property name="blockWhenExhausted" value="${jedis.blockWhenExhausted}"**/>  
 </**bean**>  
 <**bean name="jedisPool" class="redis.clients.jedis.JedisPool" destroy-method="close"**>  
  
 <**constructor-arg name="poolConfig" ref="jedisPoolConfig"**></**constructor-arg**>  
 <**constructor-arg name="host" value="${jedis.host}"**></**constructor-arg**>  
 <**constructor-arg name="port" value="${jedis.port}"**></**constructor-arg**>  
  
 </**bean**>  
 *<!-- Spring-redis连接池管理工厂 -->* <**bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"  
 p:host-name="${jedis.host}" p:port="${jedis.port}" p:password="${jedis.pass}"  
 p:pool-config-ref="jedisPoolConfig"**/>  
 *<!-- 使用中间类解决RedisCache.jedisConnectionFactory的静态注入，从而使MyBatis实现第三方缓存 -->* <**bean id="redisCacheTransfer" class="cn.cwx521.coom.RedisCacheTransfer"**>  
 <**property name="jedisConnectionFactory" ref="jedisConnectionFactory"**/>  
 </**bean**>  
</**beans**>

Mapper配置

*<!--缓存包-->*<**cache eviction="LRU" type="cn.cwx521.coom.RedisCache"** />

