# Mybatis二级缓存应用Redis预研

# 1、预研任务介绍和预研目标

## 1.1 预研任务介绍

Redis作为Mybatis的二级缓存的配置、实现方式以及实现原理，替换之后对现有框架的影响，对业务逻辑的影响，以及工作量等。

Redis作为业务缓存的更改，对现有逻辑是否有大的变动，对现有代码是否有大的更改，以及任务量和所需时间。

## 1.2 预研目标

实现Redis作为Mybatis二级缓存的配置、实现方式及原理，做出配置详解说明，确定此方法的可行性，确定工作量及工作时长等

# 2、二级缓存介绍及实现操作步骤

## 2.1 二级缓存介绍

### 2.1.1 二级缓存引入原因

通常为了减轻数据库的压力，我们会引入缓存。在Dao查询数据库之前，先去缓存中找是否有要找的数据，如果有则用缓存中的数据即可，就不用查询数据库了。 如果没有才去数据库中查找。这样就能分担一下数据库的压力。另外，为了让缓存中的数据与数据库同步，我们应该在该数据发生变化的地方加入更新缓存的逻辑代 码。这样无形之中增加了工作量，同时也是一种对原有代码的入侵。这对于有着代码洁癖的程序员来说，无疑是一种伤害。

MyBatis框架早就考虑到了这些问题，因此MyBatis提供了自定义的二级缓存概念，方便引入我们自己的缓存机制，而不用更改原有的业务逻辑。下面就让我们了解一下MyBatis的缓存机制。

### 2.1.2 缓存概述

正如大多数持久层框架一样，MyBatis 同样提供了一级缓存和二级缓存的支持；

一级缓存基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该Session中的所有 Cache 就将清空。

二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache、Hazelcast等。

一般缓存框架的数据结构基本上都是 Key-Value 方式存储，MyBatis 对于其 Key 的生成采取规则为：

[ hashcode : checksum : mappedStementId : offset : limit : executeSql : queryParams]。

对于并发 Read/Write 时缓存数据的同步问题，MyBatis 默认基于 JDK/concurrent中的ReadWriteLock，使用 ReentrantReadWriteLock 的实现，从而通过 Lock 机制防止在并发 Write Cache 过程中线程安全问题

## 2.2 Redis作为Mybatis二级缓存实现步骤

### 2.2.1 jar准备

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.5.1</version>

</dependency>

<dependency>

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

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

<version>1.4.2.RELEASE</version>

</dependency>

### 2.2.2 配置详情

解析器：结合mybatis-spring框架，读取spring关于mybatis的配置文件。具体看是否开启缓存（这里指二级缓存），如果开启，生成的执行器为CachingExecutor。

<!—开启二级缓存 -->

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

<bean id="jedisPool" class="redis.clients.jedis.JedisPool">

<constructor-arg index="0" ref="jedisPoolConfig" />

<constructor-arg index="1" value="${redis.host}" />

<constructor-arg index="2" value="${redis.port}" type="int" />

</bean>

<!-- Spring-redis连接池管理工厂 -->

<bean id="jedisConnectionFactory"

class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">

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

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

<property name="poolConfig" ref="jedisPoolConfig" />

</bean>

### 2.2.3 Redis的Cache设计

添加实现Cache接口的实现类。重写方法会在查询数据库前后调用，查询、更新、删除、创建缓存需要在这几个方法中实现。值得注意的是，getObject方法，当返回的是null时，就会接着查询。如果不为null，则返回，不再查询了。

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.RedisConnectionFailureException;

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 com.windliven.spoc.common.utils.SpringContextHolder;

public class RedisCache implements Cache {

private static final Logger logger = LoggerFactory.getLogger(RedisCache.class);

// private static JedisConnectionFactory jedisConnectionFactory = SpringContextHolder

// .getBean("jedisConnectionFactory");

private static JedisConnectionFactory jedisConnectionFactory;

private final String id;

/\*\*

\* The {@code ReadWriteLock}.

\*/

private final ReadWriteLock readWriteLock = new ReentrantReadWriteLock();

public static JedisConnectionFactory getJedisConnectionFactory() {

if (null == jedisConnectionFactory) {

jedisConnectionFactory = SpringContextHolder.getBean(JedisConnectionFactory.class);

}

return jedisConnectionFactory;

}

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 = getJedisConnectionFactory().getConnection(); // 连接清除数据

connection.flushDb();

connection.flushAll();

} catch (RedisConnectionFailureException e) {

logger.error(e.getMessage());

} 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 = getJedisConnectionFactory().getConnection();

RedisSerializer<Object> serializer = new JdkSerializationRedisSerializer(); // 借用spring\_data\_redis.jar中的JdkSerializationRedisSerializer.class

result = serializer.deserialize(connection.get(serializer.serialize(key))); // 利用其反序列化方法获取值

} catch (RedisConnectionFailureException e) {

logger.error(e.getMessage());

} 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 = getJedisConnectionFactory().getConnection();

result = Integer.valueOf(connection.dbSize().toString());

} catch (RedisConnectionFailureException e) {

logger.error(e.getMessage());

} finally {

if (connection != null) {

connection.close();

}

}

return result;

}

@Override

public void putObject(Object key, Object value) {

JedisConnection connection = null;

try {

logger.info(">>>>>>>>>>>>>>>>>>>>>>>>putObject:" + key + "=" + value);

connection = getJedisConnectionFactory().getConnection();

RedisSerializer<Object> serializer = new JdkSerializationRedisSerializer(); // 借用spring\_data\_redis.jar中的JdkSerializationRedisSerializer.class

connection.set(serializer.serialize(key), serializer.serialize(value)); // 利用其序列化方法将数据写入redis服务的缓存中

} catch (RedisConnectionFailureException e) {

logger.error(e.getMessage());

} finally {

if (connection != null) {

connection.close();

}

}

}

@Override

public Object removeObject(Object key) {

JedisConnection connection = null;

Object result = null;

try {

connection = getJedisConnectio nFactory().getConnection();

RedisSerializer<Object> serializer = new JdkSerializationRedisSerializer();

result = connection.expire(serializer.serialize(key), 0);

} catch (RedisConnectionFailureException e) {

logger.error(e.getMessage());

} finally {

if (connection != null) {

connection.close();

}

}

return result;

}

}

### 2.2.4 redis二级缓存应用

在相应的Mapper文件中，加入Cache节点，将自定义的cache实现类加进去。即：

<cache type=*"com.windliven.spoc.core.redis.RedisCache"*/>

加入该节点后就可以实现redis缓存了

# 3、总结

## 3.1 问题

(1) 使用redis缓存的时候，如果redis服务出现意外宕机情况时，服务重启过后后出现缓存数据丢失或者和业务数据库的数据不一致的情况。

(2) 并发访问时会出现redis服务宕机的情况

(3)

## 3.2 结论

(1) Mybatis二级缓存应用Redis可行，对现有业务代码改动不大，不会影响现有业务逻辑，但是必须要经过多方面测试之后才能真正应用于线上。

(2) 目前并不是项目中所有的Mapper都需要缓存，其中需要的有UserDao.xml、OfficeDao.xml、RoleDao.xml、RoleGroupDao.xml、MenuDao.xml、

ResourceCategoryDao.xml、ResourceTagDao.xml。因为这些域下的post操作都是不频繁的，所以可以缓存。

(3) 当Mybatis二级缓存应用Redis时，业务系统里面自己封装的Redis缓存的接口就被二级缓存替代了，当然，有一些特殊的数据需要缓存的时候就可以用业务系统中自己封装的接口