# Spring Boot + Mybatis + Redis 实现二级缓存

## 为什么考虑缓存机制

### 原因分析：

频繁的数据库操作是非常耗费性能的（主要是因为对于DB而言，数据是持久化在磁盘中的，因此查询操作需要通过IO，IO操作速度相比内存操作速度慢了好几个量级），尤其是对于一些相同的查询语句，完全可以把查询结果存储起来，下次查询同样的内容的时候直接从内存中获取数据即可，这样在某些场景下可以大大提升查询效率。

## Mybatis缓存

### 一级缓存

1. 一级缓存是SqlSession级别的缓存，程序建立一次数据库会话，在会话结束前，对于相同的查询，第一次查询数据库得到结果，之后会查询缓存得到结果。
2. 一级缓存是一个粗粒度的缓存，没有更新缓存和缓存过期的概念，一般而言SqlSession的生存时间很短，执行完缓存也就没有了。

### 二级缓存

1. MyBatis二级缓存的生命周期即整个应用的生命周期，应用不结束，定义的二级缓存都会存在在内存中，为了避免MyBatis二级缓存中数据量过大导致内存溢出，MyBatis在配置文件中给我们增加了很多配置例如size（缓存大小）、flushInterval（缓存清理时间间隔）、eviction（数据淘汰算法）来保证缓存中存储的数据不至于太过庞大。
2. 假如定义了MyBatis二级缓存，那么MyBatis二级缓存读取优先级高于MyBatis一级缓存。使用顺序：二级缓存 > 一级缓存 > 数据库
3. MyBatis对二级缓存的支持粒度很细，可以指定某一条查询语句是否使用二级缓存。

### 二级缓存实现

1. MyBatis对二级缓存的设计非常灵活，它自己内部实现了一系列的Cache缓存实现类，并提供了各种缓存刷新策略如LRU，FIFO等等；另外，MyBatis还允许用户自定义Cache接口实现，用户是需要实现org.apache.ibatis.cache.Cache接口，然后将Cache实现类配置在<cache type="">节点的type属性上即可；除此之外，MyBatis还支持跟第三方内存缓存库如Memecached的集成.
2. 缓存刷新策略：
   1. LRU：（Least Recently Used）,最近最少使用算法，即如果缓存中容量已经满了，会将缓存中最近做少被使用的缓存记录清除掉，然后添加新的记录
   2. FIFO：（First in first out）,先进先出算法，如果缓存中的容量已经满了，那么会将最先进入缓存中的数据清除掉
   3. Scheduled：指定时间间隔清空算法，该算法会以指定的某一个时间间隔将Cache缓存中的数据清空
3. 使用MyBatis的二级缓存有三个选择:
   1. MyBatis自身提供的缓存实现；
   2. 用户自定义的Cache接口实现；
   3. 跟第三方内存缓存库的集成，本次分析就是使用第三方缓存库Redis实现Mybatis二级缓存。

### 集成Redis

## 在pom.xml文件中引入spring-boot-starter-data-redis库

## 配置Redis，在SpringBoot项目application.yml文件中加入一下配置：

spring:

redis:

# redis数据库索引（默认为0），我们使用索引为3的数据库，避免和其他数据库冲突

database: 3

# redis服务器地址（默认为localhost）

host: localhost

# redis端口（默认为6379）

port: 6379

# redis访问密码（默认为空）

password:

# redis连接超时时间（单位为毫秒）

timeout: 0

# redis连接池配置

pool:

# 最大可用连接数（默认为8，负数表示无限）

max-active: 8

# 最大空闲连接数（默认为8，负数表示无限）

max-idle: 8

# 最小空闲连接数（默认为0，该值只有为正数才有作用）

min-idle: 0

# 从连接池中获取连接最大等待时间（默认为-1，单位为毫秒，负数表示无限）

max-wait: -1

## 实现Mybatis的二级缓存很简单，只需要新建一个类实现org.apache.ibatis.cache.Cache：

该接口共有以下五个方法：

* 1. String getId()：mybatis缓存操作对象的标识符。一个mapper对应一个mybatis的缓存操作对象。
  2. void putObject(Object key, Object value)：将查询结果塞入缓存。
  3. Object getObject(Object key)：从缓存中获取被缓存的查询结果。
  4. Object removeObject(Object key)：从缓存中删除对应的key、value。只有在回滚时触发。一般我们也可以不用实现。
  5. void clear()：发生更新时，清除缓存。
  6. int getSize()：可选实现。返回缓存的数量。
  7. ReadWriteLock getReadWriteLock()：可选实现。用于实现原子性的缓存操作。

1. 实现org.apache.ibatis.cache.Cache接口的几个关键点：
   1. 自己实现的二级缓存，必须要有一个带id的构造函数，否则会报错。
   2. 我们使用Spring封装的redisTemplate来操作Redis。也可以使用jedis库操作redis，但是jedis没有redisTemplate简介；而且，redisTemplate封装了底层的实现，未来如果我们不用jedis了，我们可以直接更换底层的库，而不用修改上层的代码；更方便的是，使用redisTemplate，我们不用关心redis连接的释放问题，否则很容易忘记释放连接而导致应用卡死。

## 需要注意的是，这里不能通过autowire的方式引用redisTemplate，因为RedisCache并不是Spring容器里的bean。所以我们需要手动地去调用容器的getBean方法来拿到这个bean，具体的实现方式看代码

## 此次分享的例子采用的redis序列化方式是默认的jdk序列化。所以数据库的查询对象（比如Product类）需要实现Serializable接口。

## 问题：

## Redis 集群如何扩展

## Redis key 如何生成的

## 数据一致性的问题：比如数据库新增成功，redis中新增失败还是以前的缓存数据

### 2011140657:-97389042:com.demo.redis.dao.mapper.ProductMapper.select:0:2147483647:SELECT \* FROM products WHERE id=? LIMIT 1:1:SqlSessionFactoryBean