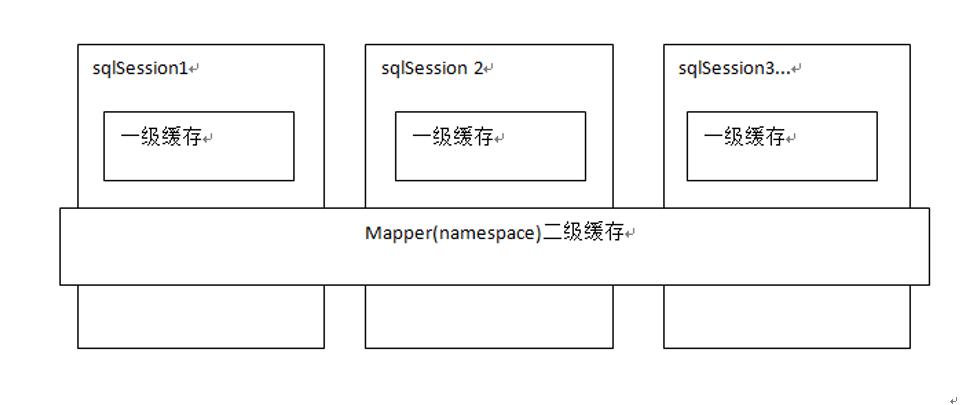
## 查询缓存

### 什么是查询缓存？

mybatis提供查询缓存，用于减轻数据压力，提高数据库性能。

mybaits提供一级缓存，和二级缓存。



* 一级缓存是SqlSession级别的缓存。在操作数据库时需要构造 sqlSession对象，在对象中有一个数据结构（HashMap）用于存储缓存数据。不同的sqlSession之间的缓存数据区域（HashMap）是互相不影响的。

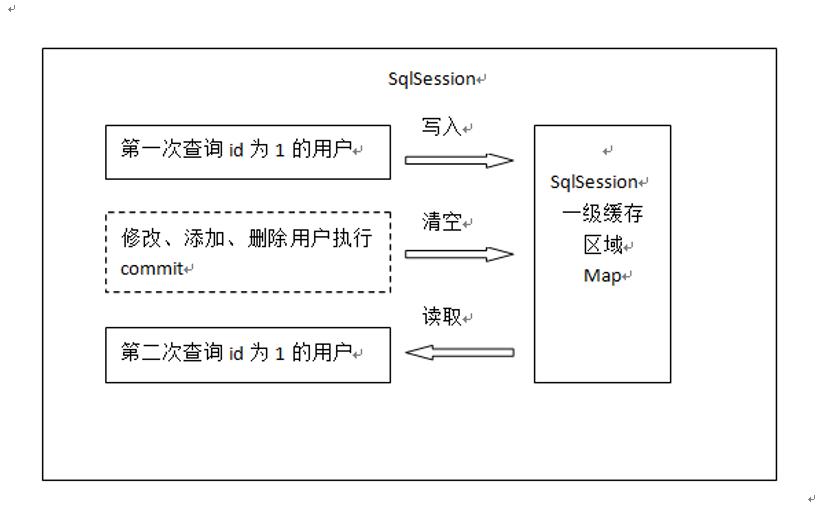
二级缓存是mapper级别的缓存，多个SqlSession去操作同一个Mapper的sql语句，多个SqlSession可以共用二级缓存，二级缓存是跨SqlSession的。

为什么要用缓存？

如果缓存中有数据就不用从数据库中获取，大大提高系统性能。

### 一级缓存

**工作原理**：



第一次发起查询用户id为1的用户信息，先去找缓存中是否有id为1的用户信息，如果没有，从数据库查询用户信息。

得到用户信息，将用户信息存储到一级缓存中。

如果sqlSession去执行commit操作（执行插入、更新、删除），清空SqlSession中的一级缓存，这样做的目的为了让缓存中存储的是最新的信息，避免脏读。

第二次发起查询用户id为1的用户信息，先去找缓存中是否有id为1的用户信息，缓存中有，直接从缓存中获取用户信息。

一级缓存应用

正式开发，是将 mybatis 和 spring 进行整合开发，事务控制在 service 中。

一个 service 方法中包括很多 mapper 方法调用。

service{

//开始执行时，开启事务，创建SqlSession对象

//第一次调用mapper的方法findUserById(1)

//第二次调用mapper的方法findUserById(1)，从一级缓存中取数据

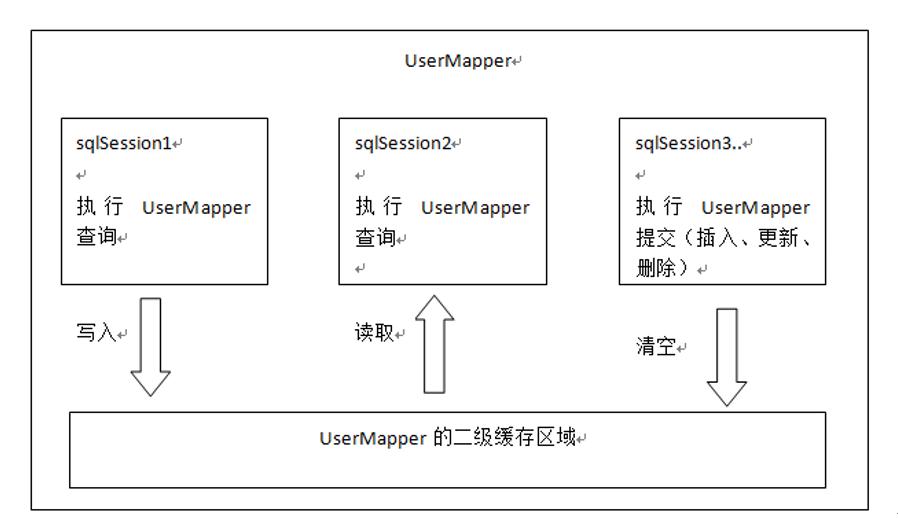
//方法结束，sqlSession关闭

}

如果是执行两次service调用查询相同的用户信息，不走一级缓存，因为session方法结束，sqlSession就关闭，一级缓存就清空。

### 二级缓存

原理



首先开启mybatis的二级缓存。

sqlSession1去查询用户id为1的用户信息，查询到用户信息会将查询数据存储到二级缓存中。

如果SqlSession3去执行相同 mapper下sql，执行commit提交，清空该 mapper下的二级缓存区域的数据。

sqlSession2去查询用户id为1的用户信息，去缓存中找是否存在数据，如果存在直接从缓存中取出数据。

二级缓存与一级缓存区别，二级缓存的范围更大，多个sqlSession可以共享一个UserMapper的二级缓存区域。

UserMapper有一个二级缓存区域（按namespace分） ，其它mapper也有自己的二级缓存区域（按namespace分）。

每一个namespace的mapper都有一个二缓存区域，两个mapper的namespace如果相同，这两个mapper执行sql查询到数据将存在相同的二级缓存区域中。

**开启二级缓存**：

mybaits的二级缓存是mapper范围级别，除了在SqlMapConfig.xml设置二级缓存的总开关，还要在具体的mapper.xml中开启二级缓存

在 SqlMapConfig.xml 开启二级开关

<!-- 开启二级缓存 -->

<setting name="cacheEnabled" value="true"/>然后在你的 Mapper 映射文件中添加一行： ，表示此 mapper 开启二级缓存。

**useCache 配置**

在 statement 中设置 useCache=false 可以禁用当前 select 语句的二级缓存，即每次查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。

<select id="findUserById" parameterType="int" resultType="user" useCache="false">

总结：针对每次查询都需要最新的数据sql，要设置成useCache=false，禁用二级缓存。

**刷新缓存（清空缓存）**

在mapper的同一个namespace中，如果有其它insert、update、delete操作数据后需要刷新缓存，如果不执行刷新缓存会出现脏读。

设置statement配置中的flushCache=”true” 属性，默认情况下为true即刷新缓存，如果改成false则不会刷新。使用缓存时如果手动修改数据库表中的查询数据会出现脏读。

如下：

<insert id="insetrUser" parameterType="cn.zhisheng.mybatis.po.User" flushCache="true">

一般下执行完commit操作都需要刷新缓存，flushCache=true表示刷新缓存，这样可以避免数据库脏读。

**Mybatis Cache参数**

flushInterval（刷新间隔）可以被设置为任意的正整数，而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新。

size（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

readOnly（只读）属性可以被设置为true或false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是false。

如下例子：

<cache eviction="FIFO" flushInterval="60000" size="512" readOnly="true"/>

这个更高级的配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会导致冲突。可用的收回策略有, 默认的是 LRU:

LRU – 最近最少使用的:移除最长时间不被使用的对象。

FIFO – 先进先出:按对象进入缓存的顺序来移除它们。

SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。

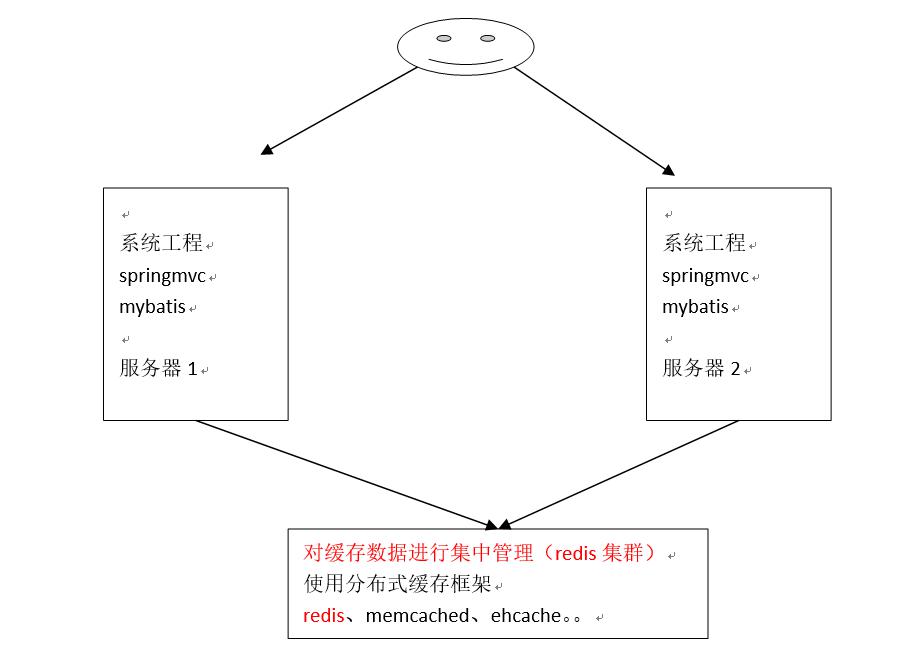
WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

**Mybatis 整合 ehcache**

ehcache 是一个分布式缓存框架。

分布缓存

我们系统为了提高系统并发，性能、一般对系统进行分布式部署（集群部署方式）



不使用分布缓存，缓存的数据在各各服务单独存储，不方便系统 开发。所以要使用分布式缓存对缓存数据进行集中管理。

mybatis无法实现分布式缓存，需要和其它分布式缓存框架进行整合。

**整合方法**

mybatis 提供了一个二级缓存 cache 接口（org.apache.ibatis.cache 下的 Cache），如果要实现自己的缓存逻辑，实现cache接口开发即可。

import java.util.concurrent.locks.ReadWriteLock;

public interface Cache {

String getId();

void putObject(Object var1, Object var2);

Object getObject(Object var1);

Object removeObject(Object var1);

void clear();

int getSize();

ReadWriteLock getReadWriteLock();

}

mybatis和ehcache整合，mybatis 和 ehcache 整合包中提供了一个 cache 接口的实现类(org.apache.ibatis.cache.impl 下的 PerpetualCache)。

package org.apache.ibatis.cache.impl;

import java.util.HashMap;

import java.util.Map;

import java.util.concurrent.locks.ReadWriteLock;

import org.apache.ibatis.cache.Cache;

import org.apache.ibatis.cache.CacheException;

public class PerpetualCache implements Cache {

private String id;

private Map<Object, Object> cache = new HashMap();

public PerpetualCache(String id) {

this.id = id;

}

public String getId() {

return this.id;

}

public int getSize() {

return this.cache.size();

}

public void putObject(Object key, Object value) {

this.cache.put(key, value);

}

public Object getObject(Object key) {

return this.cache.get(key);

}

public Object removeObject(Object key) {

return this.cache.remove(key);

}

public void clear() {

this.cache.clear();

}

public ReadWriteLock getReadWriteLock() {

return null;

}

public boolean equals(Object o) {

if(this.getId() == null) {

throw new CacheException("Cache instances require an ID.");

} else if(this == o) {

return true;

} else if(!(o instanceof Cache)) {

return false;

} else {

Cache otherCache = (Cache)o;

return this.getId().equals(otherCache.getId());

}

}

public int hashCode() {

if(this.getId() == null) {

throw new CacheException("Cache instances require an ID.");

} else {

return this.getId().hashCode();

}

}

}

通过实现 Cache 接口可以实现 mybatis 缓存数据通过其它缓存数据库整合，mybatis 的特长是sql操作，缓存数据的管理不是 mybatis 的特长，为了提高缓存的性能将 mybatis 和第三方的缓存数据库整合，比如 ehcache、memcache、Redis等。

应用场景

对于访问多的查询请求且用户对查询结果实时性要求不高，此时可采用 mybatis 二级缓存技术降低数据库访问量，提高访问速度，业务场景比如：耗时较高的统计分析sql、电话账单查询sql等。

实现方法如下：通过设置刷新间隔时间，由 mybatis 每隔一段时间自动清空缓存，根据数据变化频率设置缓存刷新间隔 flushInterval，比如设置为30分钟、60分钟、24小时等，根据需求而定。

局限性

mybatis 二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用 mybatis 的二级缓存就无法实现当一个商品变化时只刷新该商品的缓存信息而不刷新其它商品的信息，因为 mybaits 的二级缓存区域以 mapper 为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空。解决此类问题需要在业务层根据需求对数据有针对性缓存。

## 1.JDBC编程有哪些不足之处，MyBatis是如何解决这些问题的？

① 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

解决：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

② Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

③ 向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

解决： Mybatis自动将java对象映射至sql语句。

④ 对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

解决：Mybatis自动将sql执行结果映射至java对象。

## 2.MyBatis编程步骤是什么样的？

① 创建SqlSessionFactory   
② 通过SqlSessionFactory创建SqlSession   
③ 通过sqlsession执行数据库操作   
④ 调用session.commit()提交事务   
⑤ 调用session.close()关闭会话