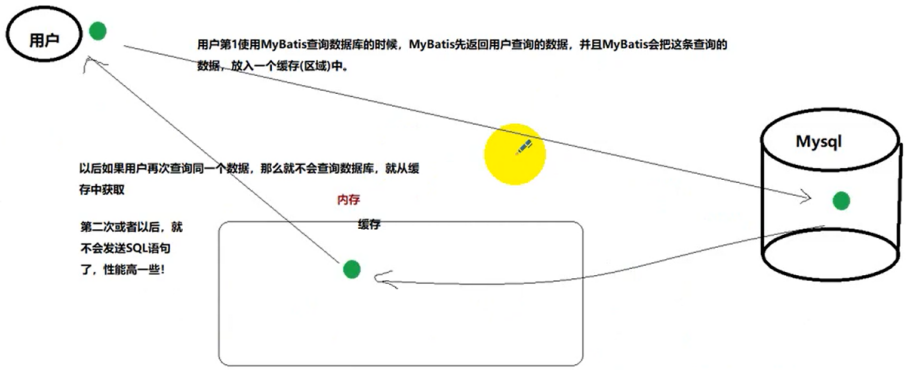
1.MyBatis缓存(面试题)

Cache 缓存



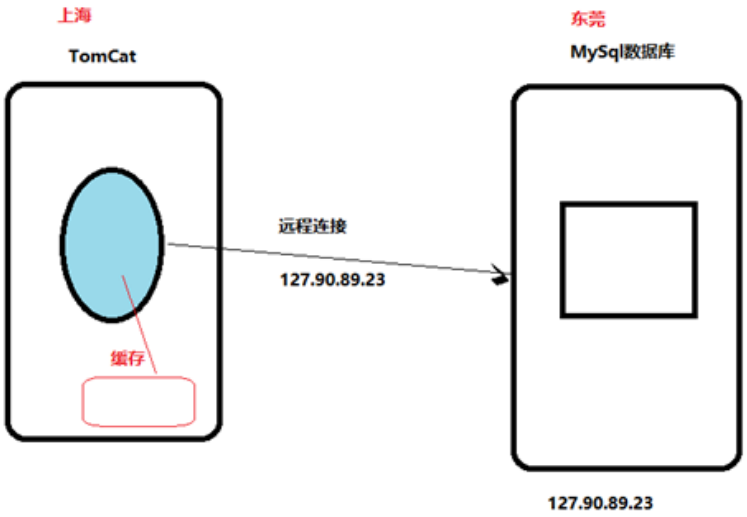
缓存一致性问题！

缓存中有，先查询缓存。缓存中没有，那么查询数据库。这样的话不用每次都查询数据库。减轻数据库的压力。提高查询效率！！！

第一次查询的时候，由于缓存中没有，那么去查询数据库返回给客户端。同时还会把这个次查询的数据放入缓存。

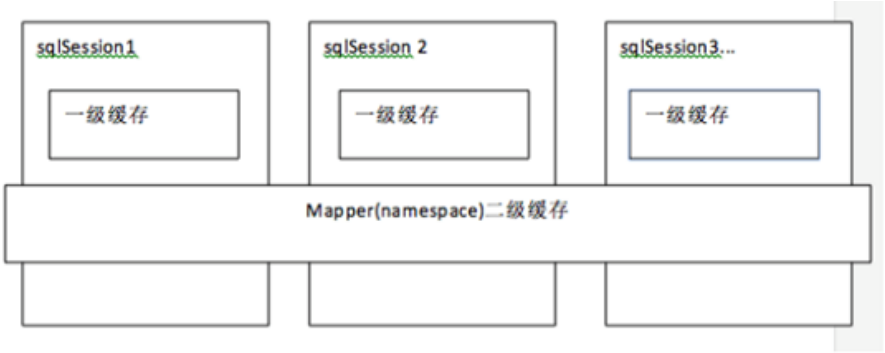
第二次查询同样的数据时候，发现缓存中曾经有查询过的数据，那么直接从缓存中读取。不必再次查询数据库，**减轻数据库服务器压力**，缓存中有就查缓存，缓存中没有就查数据库！

如果数据库中数据发生了修改，那么缓存就会清空，保持数据的一致性！防止脏数据！



1.1MyBatis缓存分析

mybatis提供查询缓存，如果缓存中有数据就不用从数据库中获取，用于减轻数据压力，提高系统性能。



Mybatis的缓存，包括一级缓存和二级缓存

1.1.1 一级缓存

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。

一级缓存指的就是sqlsession，在sqlsession中有一个数据区域，是map结构，这个区域就是一级缓存区域。一级缓存中的key是由sql语句、条件、statement等信息组成一个**唯一值**。一级缓存中的value，就是查询出的结果对象。

一级缓存是session级别的，同一个session！ 一级缓存是系统自带，是默认使用的，不需要手动开启！

1.1.2 二级缓存

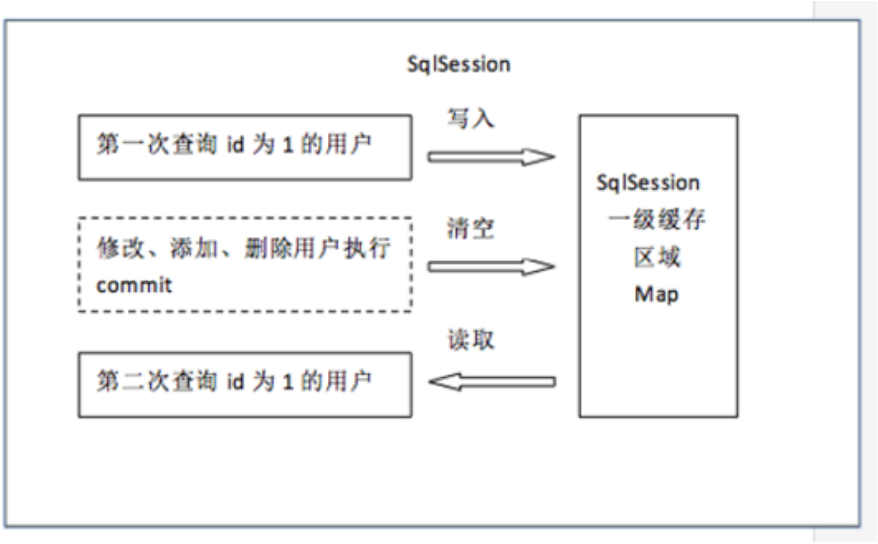
二级缓存是mapper级别的缓存，多个SqlSession去操作同一个Mapper（映射文件）的sql语句（通过调用SqlSession类中的方法，在Mybatis基础应用.md中3.2.4节搭建数据访问层的时候可以看到SqlSession调用它的方法），多个SqlSession可以共用二级缓存，二级缓存是跨SqlSession的（看了Mybatis基础应用.md中3.2节可以更清楚的理解二级缓存）。

二级缓存指的就是同一个**namespace下的mapper**，二级缓存中，也有一个map结构，这个区域就是二级缓存区域。二级缓存中的key是由sql语句、条件、statement等信息组成一个唯一值。二级缓存中的value，就是查询出的结果对象。

二级缓存，可以跨session！二级缓存是要配置，然后手动开启！

一级、二级缓存的结构：
Map<String,Object>
key 缓存标志 （唯一）
Value 缓存的数据

1.2 一级缓存



上图的解释：

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），就会自动清空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，如果没有就等同于第一次查询该信息。

Mybatis默认支持一级缓存。

- 测试1

```
@Test
public void test1(){
    Student s1 = mapper.selectOneStudent(1);
    Student s2 = mapper.selectOneStudent(1);
    System.out.println(s1==s2);
}
```

输出的结果为：true，s2变量指向的对象是s1指向的缓存中（SqlSession中的一个区域）的对象，而不是使用SQL语句查询得到的Mybatis新创建的Student对象。

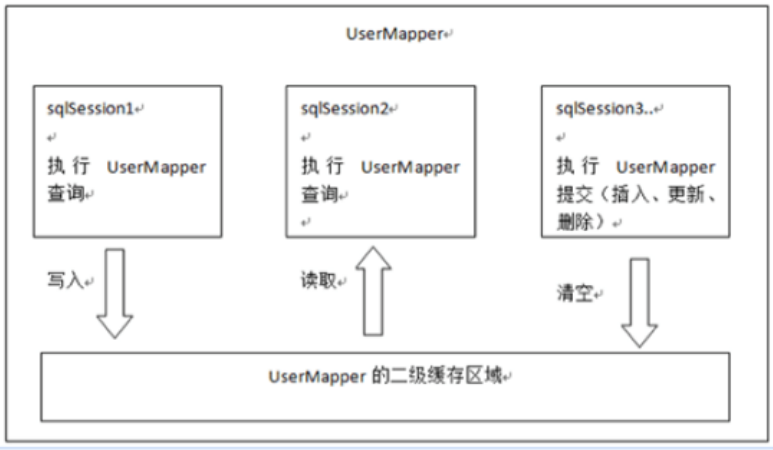
- 测试2

```
@Test
public void test1(){
    Student s1 = mapper.selectOneStudent(1);

    //清除缓存
    //session.commit();
    session.clearCache();

    Student s2 = mapper.selectOneStudent(1);
    System.out.println(s1==s2);
}
```

输出的结果为：false，s2指向的对象是第二次执行SQL语句查询的得到的结果被Mybatis重新封装成一个Student对象，因为sqlSession中的缓存已经被清除了。

1.3二级缓存

- 原理
  下图是多个sqlSession请求UserMapper的二级缓存图解。



上图的解释：（二级缓存开启的前提下）

第一次调用mapper下的SQL去查询用户信息。查询到的信息会存到该mapper对应的二级缓存区域内。

第二次调用相同namespace下的mapper映射文件（xml）中相同的SQL去查询用户信息。会去对应的二级缓存内取结果。

如果调用相同namespace下的mapper映射文件中的增删改SQL，并执行了commit操作。此时会清空该namespace下的二级缓存。

- 开启二级缓存（全局开关）步骤

  Mybatis默认是没有开启二级缓存

  a. 在核心配置文件SqlMapConfig.xml（mybatis-config.xml）中加入以下内容（开启二级缓存总开关）：
  在settings标签中添加以下内容：

```
<!-- 开启二级缓存总开关 -->
<setting name="cacheEnabled" value="true"/>
```

b. 在想要开启二级缓存的映射文件（StudentMapper）中，加入以下内容，开启二级缓存：

```
<!-- 开启本mapper下的namespace的二级缓存，默认使用的是mybatis提供的PerpetualCache -->
<cache></cache>
```

设置 cache 标签的属性

cache 标签有多个属性，一起来看一些这些属性分别代表什么意义

- `eviction`：缓存回收策略，有这几种回收策略
  - LRU - 最近最少回收，移除最长时间不被使用的对象
  - FIFO - 先进先出，按照缓存进入的顺序来移除它们
  - 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，更积极的移除基于垃圾收集器和弱引用规则的对象

默认是 LRU 最近最少回收策略

- `flushInterval` 缓存刷新间隔，缓存多长时间刷新一次，默认不清空，设置一个毫秒值
- `readOnly`：是否只读；**true 只读**，MyBatis 认为所有从缓存中获取数据的操作都是只读操作，不会修改数据。MyBatis 为了加快获取数据，直接就会将数据在缓存中的引用交给用户。不安全，速度快。**读写(默认)**：MyBatis 觉得数据可能会被修改
- `size` ：缓存存放多少个元素
- `type` ：指定自定义缓存的全类名(实现Cache 接口即可)
- `blocking` ： 若缓存中找不到对应的key，是否会一直blocking，直到有对应的数据进入缓存。

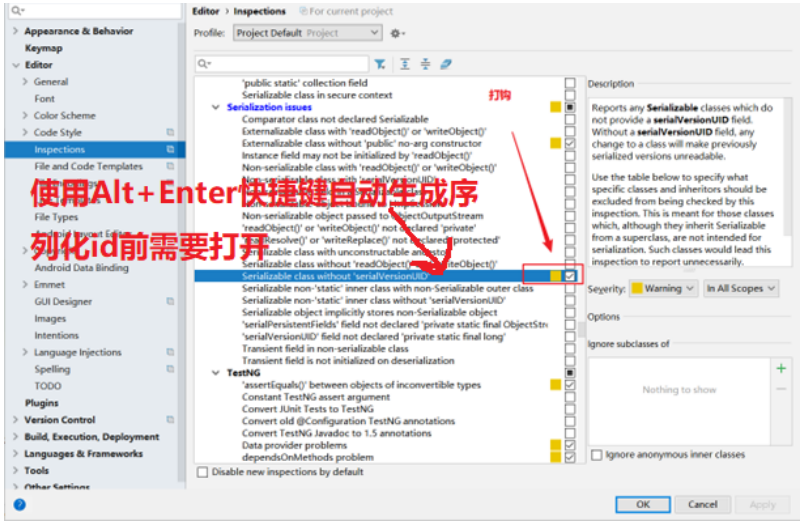
```
<!-- 配置使用二级缓存
eviction：缓存的回收策略，默认是LRU
    LRU – 最近最少使用的：移除最近最长时间不被使用的对象
    FIFO – 先进先出策略：按对象进入缓存的顺序来移除它们
    SOFT – 软引用：移除基于垃圾回收器状态和软引用规则的对象
    WEAK – 弱引用：更积极地移除基于垃圾收集器状态和弱引用规则的对象
flushInterval：缓存的刷新间隔，默认是不刷新的，单位为ms
readOnly：缓存的只读设置，默认是false
    true:只读，mybatis认为只会对缓存中的数据进行读取操作，不会有修改操作，Mybatis为了加快数据的读取，直接将缓存中对象的引用交给用户
    false：读写，mybatis认为不仅会有读取数据，还会有修改操作。会通过序列化和反序列化的技术克隆一份新的数据交给用户
size：缓存中的对象个数
type：自定义缓存或者整合第三方缓存时使用
    class MyCache implements Cache{}
bocking：缓存中没有key时，是否进行阻塞
    true：阻塞
    false：不阻塞（默认）
-->
<cache eviction="LRU"
flushInterval="60000" readOnly="false"
size="1000"></cache>
```

- 实现序列化

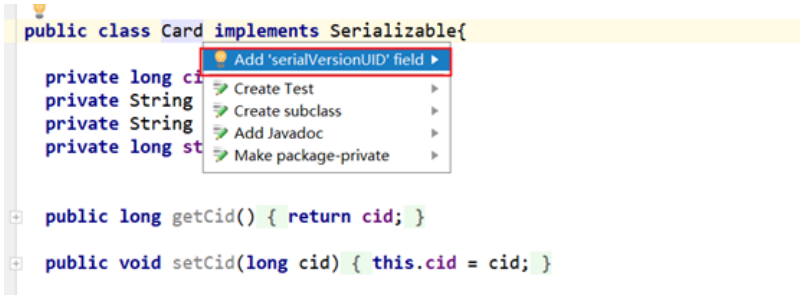
```
2019-05-09 17:02:23,446 [main] DEBUG [com.mybatis.mapper.StudentMapper] - Cache Hit Ratio [com.mybatis.mapper.StudentMapper]: 0.0
2019-05-09 17:02:23,834 [main] DEBUG [com.mybatis.mapper.StudentMapper.getStudentById1] - ==>  Preparing: SELECT * FROM student
2019-05-09 17:02:23,854 [main] DEBUG [com.mybatis.mapper.StudentMapper.getStudentById1] - ==> Parameters: 1(Integer)
2019-05-09 17:02:23,925 [main] DEBUG [com.mybatis.mapper.StudentMapper.getStudentById1] - <==      Total: 1

org.apache.ibatis.cache.CacheException: Error serializing object.  Cause: java.io.NotSerializableException: com.mybatis.bean
.Student_$$_jvstc15_0

    at org.apache.ibatis.cache.decorators.SerializedCache.serialize(SerializedCache.java:102)
    at org.apache.ibatis.cache.decorators.SerializedCache.putObject(SerializedCache.java:56)
    at org.apache.ibatis.cache.decorators.LoggingCache.putObject(LoggingCache.java:51)
    at org.apache.ibatis.cache.decorators.SynchronizedCache.putObject(SynchronizedCache.java:45)
    at org.apache.ibatis.cache.decorators.TransactionalCache.flushPendingEntries(TransactionalCache.java:122)
    at org.apache.ibatis.cache.decorators.TransactionalCache.commit(TransactionalCache.java:105)
    at org.apache.ibatis.cache.TransactionalCacheManager.commit(TransactionalCacheManager.java:44)
    at org.apache.ibatis.executor.CachingExecutor.close(CachingExecutor.java:61)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.close(DefaultSqlSession.java:264)
```

由于二级缓存的数据不一定都是存储到内存中，它的存储介质多种多样，所以需要给缓存的对象执行序列化。

缓存默认是存入内存中，但是如果需要把缓存对象存入硬盘那么久需要序列化(实体类要实现)



贯标放在Card类上，按下快捷键Alt+Enter




```
public class Card implements Serializable{

  private static final long serialVersionUID = -662505496265903627L;


  private long cid;
  private String cname;
  private String cmessage;
  private long stuId;
```

如果该类存在父类，那么父类也要实现序列化。

- 测试1

```
    @Test
    public void test2(){
        SqlSessionFactory factory =
MyBatisUtil.getSqlSessionFactory();
        SqlSession session1 =
factory.openSession();
        StudentMapper mapper1 =
session1.getMapper(StudentMapper.class);
```

```
        Student s1 = mapper1.selectOneStudent(1);
        System.out.println(s1);
        session1.close();

        SqlSession session2 =
factory.openSession();
        StudentMapper mapper2 =
session2.getMapper(StudentMapper.class);
        Student s2 = mapper2.selectOneStudent(1);
        System.out.println(s2);
    }
```

如果第一个session没有提交，则没有进行二级缓存。

所以，想实现二级缓存，需要前面的session已经提交过，并且相同的提交sql。

注意：如果在测试中打印session1 == session2，那么得到的结果是false，这两不是同一个对象是在映射文件中使用的catch标签中属性readOnly的值为false，修改这个属性值为true，则这里得到的结果为true；
输出的结果中有
`Cache Hit Ratio [com.bruce.mapper.UserMapper]: 0.5` 那么就是第二次查询的时候使用了二级缓存中的数据。

- 开启二级缓存的局部开关

  该statement中设置useCache=false，可以禁用当前select语句的二级缓存，即每次查询都是去数据库中查询，默认情况下是true，即该statement使用二级缓存。

```
<select id="selectOneStudent" resultMap="stu" useCache="false">
    select * from student
    where id = #{id}
</select>
```

1.4 总结

1）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，默认打开一级缓存。

2）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。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现Serializable序列化接口(可用来保存对象的状态),可在它的映射文件中配置<cache/> ；

3）对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存Namespaces)的进行了C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。