http://blog.csdn.net/u011403655/article/details/46696065

一级缓存

首先做一个**测试**，创建一个mapper配置文件和mapper接口，我这里用了最简单的查询来演示。

```
<mapper namespace="cn.elinzhou.mybatisTest.mapper.UserMapper">
    <select id="findUsers"
resultType="cn.elinzhou.mybatisTest.pojo.User">
        SELECT * FROM user
    </select>
</mapper>
```

```
public interface UserMapper {
    List<User> findUsers()throws Exception;
}
```

然后编写一个单元测试

```
public class UserMapperTest {

    SqlSession sqlSession = null;
    @Before
    public void setUp() throws Exception {
        // 通过配置文件获取数据库连接信息
        Reader reader =
Resources.getResourceAsReader("cn/elinzhou/mybatisTest/config/mybatis.xml");
        // 通过配置信息构建一个SqlSessionFactory
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(reader);
        // 通过sqlSessionFactory打开一个数据库会话
        sqlSession = sqlSessionFactory.openSession();
    }

    @Test
    public void testFindUsers() throws Exception {
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        List<User> users = userMapper.findUsers();
        System.out.println(users);
    }
```

```
}
```

运行，可以看到控制台输出（先配好log4j）为类似如下图日志

```
2015-06-30 13:08:14,025 [main] DEBUG [cn.elinzhou.mybatisTest.mapper.UserMapper.findUsers] - ==>  Preparing: SELECT * FROM user
2015-06-30 13:08:14,129 [main] DEBUG [cn.elinzhou.mybatisTest.mapper.UserMapper.findUsers] - ==> Parameters:
2015-06-30 13:08:14,211 [main] DEBUG [cn.elinzhou.mybatisTest.mapper.UserMapper.findUsers] - <==      Total: 8
[User{address='null', id=1, name='null', birthday=null, sex=2}, User{address='北京市', id=10, name='null', birthday=Thu Jul 10 00:00:00 CST
```

日志说明了该操作执行的sql语句已经查询的内容，最后一行是我手动通过System.out.printf输出的结果。

然后再加一条语句

```
users = userMapper.findUsers();
```

之前的单元测试就变成了这个样子

```
public void testFindUsers() throws Exception {
    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
    List<User> users = userMapper.findUsers();
    users = userMapper.findUsers();
    System.out.println(users);
}
```

也就是在执行完userMapper.findUsers();后立刻再执行一遍userMapper.findUsers(); 可以想象，其实这两个操作执行的sql是完全相同的，而且在这期间没有对**数据库**进行过其他操作。然后执行该单元测试，发现效果跟上面执行一条的时候完全相同，也就是执行第二次userMapper.findUsers();操作的时候没有对数据库进行查询，那么得到的数据是从哪里来的？答案是一级缓存。

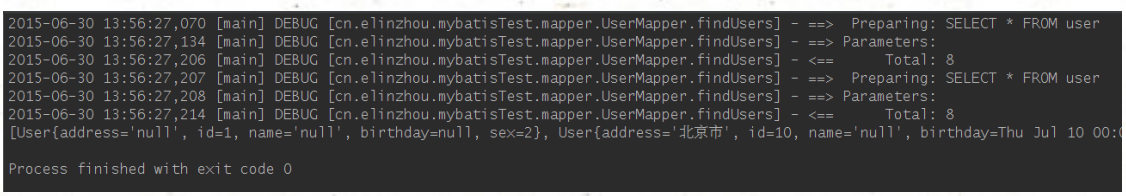
mybatis一级缓存是指在内存中开辟一块区域，用来保存用户对数据库的操作信息（sql）和数据库返回的数据，如果下一次用户再执行相同的请求，那么直接从内存中读数数据而不是从数据库读取。

其中数据的生命周期有两个影响因素。

1. 对sqlsession执行commit操作(insert/update/delete)时

对sqlsession执行commit操作，也就意味着用户执行了update、delete等操作，那么数据库中的数据势必会发生变化，如果用户请求数据仍然使用之前内存中的数据，那么将读到脏数据。所以在执行sqlsession操作后，会清除保存数据的HashMap，用户在发起查询请求时就会重新读取数据并放入一级缓存中了。

```
@Test
public void testFindUsers() throws Exception {
    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
    List<User> users = userMapper.findUsers();
    sqlSession.commit();
    users = userMapper.findUsers();
    System.out.println(users);
}
```



上述测试就是在第一查询完后执行了commit操作，再进行查询。与之前的测试不同的是，这次测试控制台打印了两组查询结果，说明在commit之后mybatis对数据重新进行了查询。

1. 关闭sqlsession

在mybatis集成**spring**时，会把SqlSessionFactory设置为单例注入到IOC容器中，不把sqlsession也设置为单例的原因是sqlsession是线程不安全的，所以不能为单例。那也就意味着其实是有关闭sqlsession的过程的。其实，对于每一个service中的sqlsession是不同的，这是通过mybatis-spring中的org.mybatis.spring.mapper.MapperScannerConfigurer创建sqlsession自动注入到service中的。(每一次增删改查后都会关闭sqlsession, 再次增删改查时重新opensisson, 关闭session后一级缓存就会失效)

而一级缓存的设计是每个sqlsession单独使用一个缓存空间，不同的sqlsession是不能互相访问数据的。当然，在sqlsession关闭后，其中数据自然被清空。

特此警告！！！！

当MyBatis与spring整合后，如果没有事务，一级缓存是失效的！一级缓存是失效的！一级缓存是失效的！

原因就是两者结合后，sqlsession如果发现当前没有事务，那么每执行一个mapper方法，sqlsession就被关闭了。如果需要维持一级缓存的可用性，有两种途径：

1. 添加事务
2. 使用二级缓存

二级缓存

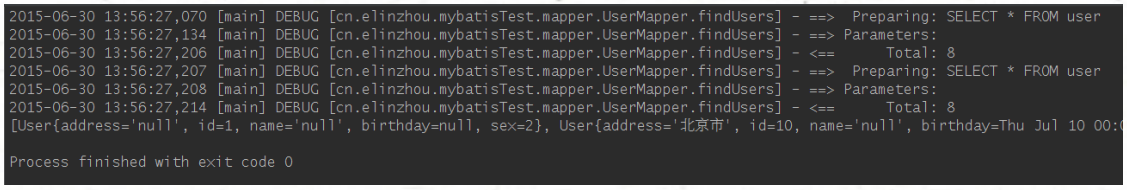
在使用二级缓存之前，先测试之前提到过的关闭sqlsession后会清空缓存的问题，把junit代码修改一下

```
@Test
    public void testFindUsers() throws Exception {
        UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
        List<User> users = userMapper.findUsers();
        //关闭sqlsession
        sqlSession.close();

        //通过sqlsessionFactroy创建一个新的sqlsession
```

```
        sqlSession = sqlSessionFactory.openSession();
        //获取mapper对象
        userMapper = sqlSession.getMapper(UserMapper.class);
         users = userMapper.findUsers();
        System.out.println(users);
    }
```

这段代码在第一次查询完后关闭sqlsession，然后创建新的sqlsession和mapper来重新执行一次查询操作，可以预见，执行结果如图



说明关闭了sqlsession后的确把之前的缓存数据清空了，之后再执行同样的查询操作也会再访问一遍数据库。为了解决这个问题，需要使用二级缓存

一级缓存的作用域仅限于一个sqlsession，但是二级缓存的作用域是一个namespace。但并不是意味着同一个namespace创建的mapper可以互相读取缓存内容，**这里的原则是，如果开启了二级缓存，那么在关闭sqlsession后，会把该sqlsession一级缓存中的数据添加到namespace的二级缓存中。**

接下测试，先需要开启二级缓存。

1.打开二级缓存总开关

打开总开关，只需要在mybatis总配置文件中加入一行设置

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

2.打开需要使用二级缓存的mapper的开关

在需要开启二级缓存的mapper.xml中加入caceh标签

```
<cache/>
```

3.POJO序列化

让需要使用二级缓存的POJO类实现Serializable接口，如

```
public class User implements Serializable {...}
```

通过之前三步操作就可以使用二级缓存了，接下来测试。添加一个Junit方法

```
@Test
    public void testFindUsersCache() throws Exception {
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        List<User> users = userMapper.findUsers();
```

```
        //关闭sqlsession
        sqlSession.close();

        //通过sqlsessionFactroy创建一个新的sqlsession
        sqlSession = sqlSessionFactory.openSession();
        //获取mapper对象
        userMapper = sqlSession.getMapper(UserMapper.class);
        //二级缓存已经开启,sqlSession1关闭后一级缓存数据添加到二级缓存,因此走了二级缓存
        users = userMapper.findUsers();
        System.out.println(users);
    }
```

执行后可以发现，控制台值输出了一次查询过程，也可以证明二级缓存开启成功。
还有一个问题，之前说了，即使开启了二级缓存，不同的sqlsession之间的缓存数据也不是想互访就能互访的，必须等到sqlsession关闭了以后，才会把其一级缓存中的数据写入二级缓存。为了测试这个，把上述代码中的

```
sqlSession.close();
```

注释，那么之前的代码就变成了

```
@Test
public void testFindUsersCache() throws Exception {
    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
    List<User> users = userMapper.findUsers();
    //关闭sqlsession
    //sqlSession.close();

    //通过sqlsessionFactroy创建一个新的sqlsession
    sqlSession = sqlSessionFactory.openSession();
    //获取mapper对象
    userMapper = sqlSession.getMapper(UserMapper.class);
    users = userMapper.findUsers();
    System.out.println(users);
}
```

再执行，发现控制台又输出了两次的查询过程，所以可以印证，**只有关闭了sqlsession之后，才会把其中一级缓存数据写入二级缓存。**

缓存配置

- 关闭刷新

在默认情况下，当sqlsession执行commit后会刷新缓存，但是也可以强制设置为不刷新，在不需要刷新的标签中加入

```
flushCache="false"
```

如

```
<select id="findUsers"
```

```
resultType="cn.elinzhou.mybatisTest.pojo.User"
flushCache="false">
```

那么，无论是否执行commit，缓存都不会刷新了。但是这样会造成脏读，只有在特殊情况下才使用

- 自动刷新

有些情况下，需要设置自动刷新缓存，那么需要配置对应mapper中的cache标签。

```
flushInterval="10000"
```

该属性表示每隔10秒钟自动刷新一遍缓存

mybatis之sqlSession无需开发人员close

http://www.cnblogs.com/langtianya/archive/2013/03/04/2942938.html

根据mybatis官方文档，建议对sqlSession进行如下操作

```
SqlSession session = sqlSessionFactory.openSession();try{// following 3 lines pseudocod for "doing some work"
    session.insert(...);
    session.update(...);
    session.delete(...);
    session.commit();}finally{
    session.close();}
```

每个操作都进行打开和关闭，而且都是重复性操作。

于是mybatis-3.0.6采用动态代理实现的aop非常好的解决了上面的问题。

大体实现方式如下：

1.在applicationContext.xml中配置

```
<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="configLocation">
            <value>classpath:mybatisConfig.xml</value>
        </property>
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
    </bean>

    <bean id="sqlSession"
```

class="org.mybatis.spring.SqlSessionTemplate">
<constructor-arg index="0" ref="sqlSessionFactory" />
</bean>

2.在具体的使用类中执行如下操作，即可。

int identifier = sqlSession.insert(statement, object);

大体原理：

执行sqlSession.insert及执行SqlSessionTemplate.insert，其内部调用sqlSessionProxy.insert，sqlSessionProxy是一个动态代理器，其调用处理器是SqlSessionInterceptor，该处理器先getSqlSession，然后执行insert，最后closeSqlSession。

不明白的同学可直接查看mybatis-3.0.6源码