# mybatis

懒加载： 用到哪个查询哪个

<**settings**>  
 *<!--懒加载-->* <**setting name="lazyLoadingEnabled" value="true"**/>

<setting name="aggressiveLazyLoading" value="false"></setting>  
 </**settings**>

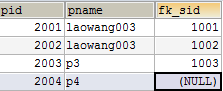
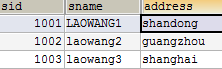
类中代码

**public class** Person **implements** Serializable {  
 **private int pid** ;  
 **private** String **pname** ;  
 **private int fk\_sid** ;  
 **private** Student **stu** ;

**public class** Student **implements** Serializable {  
 **private int sid** ;  
 **private** String **sname** ;

数据库表

person表 student表

mapper代码和配置

<**resultMap id="resultMap001" type="com.offcn.pojo.Person"**>  
 <**id property="pid" column="pid"**></**id**>  
 <**result property="pname" column="pname"**></**result**>  
 <**association property="stu"  
 javaType="com.offcn.pojo.Student"  
 column="fk\_sid"  
 select="com.offcn.mapper.StudentMapper.getStuById"**>  
 </**association**>  
</**resultMap**>  
*<!--public Person getPersonById(Integer id);-->* <**select id="getPersonById" parameterType="int" resultMap="resultMap001"**>  
 select *\** from Person where pid = #{id}  
 </**select**>

*<!-- public Student getStuById(int id);-->*<**select id="getStuById" parameterType="int" resultType="com.offcn.pojo.Student"**>  
 select *\** from student where sid = #{id}  
</**select**>

测试类：

**public void** test001() **throws** Exception {  
  
 SqlSession session = CreateSessionUtil.*getSesion*();  
 PersonMapper personMapper = session.getMapper(PersonMapper.**class**);  
  
 Person person = personMapper.getPersonById(2001);

当调用person数据的时候，Stduent的构造器不走

当调用person中的student数据的时候，student构造器才会走  
 System.***out***.println(person.getStu().getSname());  
 session.commit();  
 session.close();  
}

二级缓存：

配置在SqlMapConfig.xml中下列配置

<**settings**>  
 *<!--懒加载-->* <**setting name="lazyLoadingEnabled" value="true"**/>  
 *<!--缓存-->* <**setting name="cacheEnabled" value="true"**></**setting**>  
</**settings**>

配置在PersonMapper.xml中<**cache**></**cache**>（需要缓存的类就在相应的配置文件中加）  
 *<!--UPDATE person SET pname = 'laowang003' WHERE pid = 2002  
 public void updatePerson();  
 -->*<**update id="updatePerson"** >  
 UPDATE person SET pname = 'laowang003' WHERE pid = 2001  
</**update**>

测试类：

@Test  
 **public void** test001() **throws** Exception {  
 InputStream resourceAsStream = **null**;  
 **try** {  
 resourceAsStream = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);  
 } **catch** (IOException e) {  
 }  
 SqlSessionFactory build = **new** SqlSessionFactoryBuilder().build(resourceAsStream);  
 SqlSession session = build.openSession();  
*// 第一次* PersonMapper personMapper = session.getMapper(PersonMapper.**class**);  
 Person personById = personMapper.getPersonById(2001);  
 System.***out***.println(**"第一次查询=="**+personById.getPname());  
 session.commit();  
 session.close();  
*// 增删改 》》修改* SqlSession session3 = build.openSession();  
 PersonMapper personMapper3= session3.getMapper(PersonMapper.**class**);  
 personMapper3.updatePerson();  
 session3.commit();  
 session3.close();  
*// 第二次* SqlSession session2 = build.openSession();  
 PersonMapper personMapper2 = session2.getMapper(PersonMapper.**class**);  
 Person personById2 = personMapper2.getPersonById(2001);  
  
 System.***out***.println(**"第二次查询"**+personById2.getPname());  
 session2.commit();  
 session2.close();

SqlSession session4 = build.openSession();  
 PersonMapper personMapper4 = session4.getMapper(PersonMapper.**class**);  
 Person personById4 = personMapper4.getPersonById(2001);  
 System.***out***.println(**"第二次查询"**+personById4.getPname());  
 session4.commit();  
 session4.close();  
 }

public PageInfo<Person> getPerson(){//PageInfo就是分页  
 PageHelper.*startPage*(1,2);//分页的起始位置。和每页几条数据

List<Person> list =personService.getPerson();

<!--分页插件-->  
<property name="plugins">  
 <array>  
 <bean class="com.github.pagehelper.PageInterceptor">  
 <property name="properties">  
 <!--使用下面的方式配置参数，一行配置一个 -->  
 <value>  
 params=value1  
 </value>  
 </property>  
 </bean>  
 </array>  
</property>

**懒加载 通俗一点来说，可以比喻成懒汉模式，只有当你的对象被调用，或者对象的数据被访问的时候，才会创建对象。**

**比如你主表的外键是从表的主键，而你主表相对应的实体类中有从表的虚拟对象，那么当你查询主表的对象时时可以查询出主从俩表的数据的**

**但是当你不想要从表数据的时候，你可以使用懒加载，只加载主表的对象和数据，从表的构造并不会走，也就不会有庞大的查询压力**

**二级缓存就是mapper域的缓存**

**一级缓存是session域**

**使用条件： 1）查询的内容相同，查询的内容和以前的查询内容很相似的时候使用**

**2）如果遇到增删改，那么二级缓存会自动清空**

**比如说我查询过毛衣，那么就把他查询到的信息存到缓存里，当你下次查询毛衣的时候，不会再去创建连接sql的驱动，也不会重新开启session**

**减轻服务器压力，直接从缓存中拿到数据，但是当你对数据库进行过增删改操作后，那么二级缓存会自动清空，因为你数据的变动，我可能把毛衣改成了半袖，那么当你再去查询毛衣的时候，我数据库已经变动，这时候缓存还存储着毛衣就没有意义了，但是这次会把修改之后的半袖存进缓存中，以方便你下次查询。**

**SqlSessionFactoryBuilder**

　　SqlSessionFactoryBuilder的作用是创建SqlSessionFactory。一旦创建了SqlSessionFactory，就不再需要它了。因此SqlSessionFactoryBuilder只能存在于创建SqlSessionFactory的方法中，最佳作用域是方法作用域（也就是局部方法变量）。

**SqlSessionFactory**

　　SqlSessionFactory可以被认为是一个数据库连接池，作用是创建SqlSession接口对象。因为MyBatis的本质是使用Java语言完成数据库操作，所以SqlSessionFactory一旦被创建就会在MyBatis应用的运行期间一直存在，不能被清除或重建。因此SqlSessionFactory的最佳作用域是应用作用域。因为SqlSessionFactory占有数据库的连接资源，如果创建多个SqlSessionFactory，那么就存在多个数据库连接池，不利于控制数据库资源，并会导致数据库连接资源耗尽，所以最好把SqlSessionFactory作为一个单例，也就是使用单例模式。

**SqlSession**

如果说SqlSessionFactory相当于数据库连接池，那么SqlSession就相当于一个数据库连接（Connection对象），代表一个请求。每个线程都有自己的SqlSession实例。SqlSession实例不是线程安全的，不能被共享，所以它的最佳作用域是方法作用域（也就是局部方法变量）。把关闭操作放到finally块中以确保每次都能执行关闭。

**映射器实例（Mapper Instances）**

　　Mapper代表某个请求中的一个业务处理步骤，是一个接口，绑定了映射语句的Mapper实例是从SqlSession中获得的。它的最佳作用域是方法作用域（也就是局部方法变量），生命周期小于或等于SqlSession的生命周期。