两个持久化类Clazz和Student

```
public class Clazz implements Serializable {

    private Integer id;
    private String code;
    private String name;
    private List<Student> students;
```

```
public class Student implements Serializable {

    private Integer id;
    private String name;
    private String sex;
    private Integer age;
    private Clazz clazz;
```

注意Clazz中定义的List对象，用来映射一对多的关联关系

Student中定义的Clazz对象，用来映射一对一的关联关系（可能有人会疑问不是多对一吗？可以仔细想一想，只对每个学生本身来说和班级是不是一对一的）

对应的两个mapper

```
<mapper namespace="com.jlb.mapper.ClazzMapper">
    <select id="selectClazzById" parameterType="int" resultMap="clazzMap">
        select * from tb_clazz where id=#{id}
    </select>

    <resultMap type="com.jlb.entity.Clazz" id="clazzMap">
        <id property="id" column="id" />
        <result property="code" column="code" />
        <result property="name" column="name" />
        <!-- 一对多关联映射：collection -->
        <collection property="students" javaType="ArrayList"
            column="id" ofType="com.jlb.entity.Student"
            select="com.jlb.mapper.StudentMapper.selectStudentByClazzId"
            fetchType="lazy">
            <id property="id" column="id" />
            <result property="name" column="name" />
            <result property="sex" column="sex" />
            <result property="age" column="age" />
        </collection>
    </resultMap>
</mapper>
```

依旧老规矩，容我慢慢道来

ClazzMapper只有一个操作，根据自身id查询班级信息，不过返回结果包含了一个班级学生学生信息的list，所以还定义了一个resultMap，这次的resultMap大部分与之前的不无不同，但是这次我们要处理的是一对多的关联映射，所以这里用到了新的标签collection，这个一对多嘛，本身是不存在什么外键的说法的，我们只能通过多方存在的外键去获取数据，column指明了参数，因为使用了反向select（即从另一个mapper读取数据），所以使用了ofType，fetchType表示懒加载，操作时要获取这部分数据时才会执行，提高了效率。

```xml
<mapper namespace="com.jlb.mapper.StudentMapper">
    <!-- 根据id查询学生信息，多表连接，返回resultMap -->
    <select id="selectStudentById" parameterType="int" resultMap="studentMap">
        select * from tb_clazz c,tb_student s where c.id=s.clazz_id and
        s.id=#{id}
    </select>

    <!-- 根据班级id查询学生信息，返回resultMap -->
    <select id="selectStudentByClazzId" parameterType="int"
    resultMap="studentMap">
    select * from tb_student where clazz_id=#{id}
    </select>
    <!-- 映射Student对象的resultMap -->
    <resultMap type="com.jlb.entity.Student" id="studentMap">
        <id property="id" column="id" />
        <result property="name" column="name" />
        <result property="sex" column="sex" />
        <result property="age" column="age" />
        <!-- 多对一关联映射：association 可理解为一个学生还是只对应一个班级，所以还是association -->
        <association property="clazz" column="clazz_id"
            javaType="com.jlb.entity.Clazz">
            <id property="id" column="id" />
            <result property="code" column="code" />
            <result property="name" column="name" />
        </association>
    </resultMap>
</mapper>
```

StudentMapper有两个操作，一个根据自身id查询学生信息，一个根据班级id即外键查询这个班级所有的学生信息（为了ClazzMapper的反向select准备），谈谈resultMap，这里的association有所不同的是没有运用反向select，所以是将各属性列出来的，并且查询学生信息时是用的多表查询，这样Clazz的信息也就有了。

两个接口

```java
public interface ClazzMapper {

    Clazz selectClazzById(Integer id);
}
```

```java
public interface StudentMapper {

    Student selectStudentById(Integer id);
}
```

测试方法，一对多方向和多对一方向都有测试

```
public static void main(String[] args) {
    // 获得Session实例
    SqlSession session = SelfSqlSessionFactory.getSqlSession();

    OneToMany oneToMany = new OneToMany();
    //oneToMany.testSelectClazzById(session);
    oneToMany.testSelectStudentById(session);

    session.commit();
    session.close();

}

public void testSelectClazzById(SqlSession session) {
    ClazzMapper clazzMapper = session.getMapper(ClazzMapper.class);
    Clazz clazz = clazzMapper.selectClazzById(1);
    System.out.println(clazz.getCode() + " " + clazz.getName());
    System.out.println(clazz.getStudents().get(0).getClazz().getCode());
}

public void testSelectStudentById(SqlSession session){
    StudentMapper studentMapper=session.getMapper(StudentMapper.class);
    Student student=studentMapper.selectStudentById(1);
    System.out.println(student.getSex());
    System.out.println(student.getClazz().getCode());
}
```

同样的，这里介绍注解配置的方法，与上面对应的看

先看一方对应多方的查询

ClazzMapper

```
@Select("select * from tb_clazz where id=#{id}")
@Results({
    @Result(id=true,column="id",property="id"),
    @Result(column="code",property="code"),
    @Result(column="name",property="name"),
    @Result(column="id",property="students",
    many=@Many(
            select="com.jlb.mapper.StudentMapper.selectStudentByClazzId",
            fetchType=FetchType.LAZY))
})
Clazz selectClazzById(Integer id);
```

这里要注意的就是many=@Many和collection的区别，这俩指明参数来源，以及对应持久化类中的属性后，直接指明反向select就行了，不需要指明反向select后的属性

StudentMapper

```
@Select("select * from tb_student where clazz_id=#{id}")
@Results({
    @Result(id=true,column="id",property="id"),
    @Result(column="name",property="name"),
    @Result(column="sex",property="sex"),
    @Result(column="age",property="age")
})
List<Student> selectStudentByClazzId(Integer id);
```

@Results可写可不写，因为没有特殊的返回属性，虽然xml中也要指明返回类型，但注解这边确实更加明了

对于一对一应用笔记里所说哪些情况下还是要考虑xml，这里遇到多表查询，不通过反向select，我实在是没想通通过注解，将那个Results写出来，所以感觉这种情况还是用xml配置的好