两个持久化类Card和Person

```
public class Card implements Serializable {

    private Integer id;
    private String code;
```

```
public class Person implements Serializable {

    private Integer id;
    private String name;
    private String sex;
    private Integer age;
    private Card card;
```

注意Person类定义的Card对象，对应数据表的card_id，用来映射一对一的关联关系

对应的两个mapper

```
<mapper namespace="com.jlb.mapper.CardMapper">
    <select id="selectCardById" parameterType="int" resultType="com.jlb.entity.Card">
        select * from tb_card where id=#{id}
    </select>
</mapper>
```

```
<mapper namespace="com.jlb.mapper.PersonMapper">
    <select id="selectPersonById" parameterType="int" resultMap="personMapper">
        select * from tb_person where id=#{id}
    </select>
    <!-- 映射Person对象的resultMap -->
    <resultMap type="com.jlb.entity.Person" id="personMapper">
        <id property="id" column="id" />
        <result property="name" column="name" />
        <result property="sex" column="sex" />
        <result property="age" column="age" />
        <!-- 一对一关联映射: association -->
        <association property="card" column="card_id"
            select="com.jlb.mapper.CardMapper.selectCardById" javaType="com.jlb.entity.Card" />
    </resultMap>
</mapper>
```

慢慢道来各个属性的作用，

namespace：一来其它的关联mapper要调用本mapper的某个查询方法时需要通过该名去定位。二来这个主要的作用是绑定Dao接口，即面向接口编程，不需要写实现类，也会通过该绑定自动找到要执行的sql

id：唯一定位一个操作方法

parameterType：指明参数的类型

resultType：查询返回的类型

resultMap：存在有关联查询的时候，先将查询的结果统一封装在resultMap中再输出

association：一对一关联映射，property指明持久化类中哪个属性，column指明关联数据表中那个字段，select指明调用哪个关联mapper进行相关的查询，javaType指明此关联查询的返回类型

因为我们要实现根据一个人的id查询这个人的信息，所以对应创建了PersonMapper的接口类，如所见，类名与namespace定义的对应，同样还有的讲究就如图中注释所讲

```
public interface PersonMapper {

    //方法名与对应xml的对应select的id相同，参数与parameterType属性一致
    Person selectPersonById(Integer id);
}
```

简单的应用就是先创建session对象，再由session获取mapper接口的代理对象，通过该代理对象就可以执行相关的接口方法，通过绑定自动找到要执行的sql

```
        // 获得Session实例
        SqlSession session = SelfSqlSessionFactory.getSqlSession();

        //获得mapper接口的代理对象
        PersonMapper pm=session.getMapper(PersonMapper.class);

        Person p=pm.selectPersonById(1);

        System.out.println(p.getName());
        System.out.println(p.getCard().getCode());

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

上面介绍完了通过xml操作，下面对应的用注解的方式实现上述操作，同样的东西就不复述了，将区别罗列出来就一目了然了

首先呢，既然使用了注解配置，那么对应的mapper的xml文件都不需要了，但对应的，不管有没有实际业务需求的方法，每个持久化类都得有个mapper接口

得注意mybatis配置文件这里的区别

```
<!-- mappers告诉mybatis去哪找持久化类的映射文件 -->
<mappers>
    <mapper resource="com/jlb/mapper/CardMapper.xml" />
    <mapper resource="com/jlb/mapper/PersonMapper.xml" />
</mappers>
```

```
 <!-- mappers告诉mybatis去哪找持久化类的映射文件 -->
 <mappers>
     <mapper class="com.jlb.mapper.CardMapper" />
     <mapper class="com.jlb.mapper.PersonMapper" />
 </mappers>
```

一目了然

然后我来慢慢介绍接口

CardMapper

```
public interface CardMapper {

    @Select("select * from tb_card where id=#{id}")
    Card selectCardById(Integer id);
}
```

和上面xml中的操作对应看，其实差不多，@Select表明是查询，里面写好查询语句，我们可以看出比之xml配置确实简洁不少，不需要设置参数类型，也不需要设置返回类型

PersonMapper

```
public interface PersonMapper {

    @Select("select * from tb_person where 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"),
            @Result(column="card_id",property="card",
            one=@One(select="com.jlb.mapper.CardMapper.selectCardById",fetchType=FetchType.EAGER)
                    )
    })
    Person selectPersonById(Integer id);

}
```

依旧和上面xml中的操作对应看，因为这边有一对一的关联映射，在xml得配置一个ResultMap作为返回类型，这里同样不需要设置返回类型，只不过多了一个@Results配置，仔细观察里面的@Result，其实就和xml中ResultMap下的子标签对

应，不过在处理一对一关联映射时没有再使用association，而是使用反向select时，使用了one=@One，FetchType.EAGER表示查询立即加载，和懒加载对应

就这样，结果和使用xml一样，我们可以清楚地观察到，使用注解确实比xml方便了不止一点半点，但据大神说，有些场合还是xml要好，目前暂时体会不到，等体会到再说。