# MyBatis

内部封装了JDBC

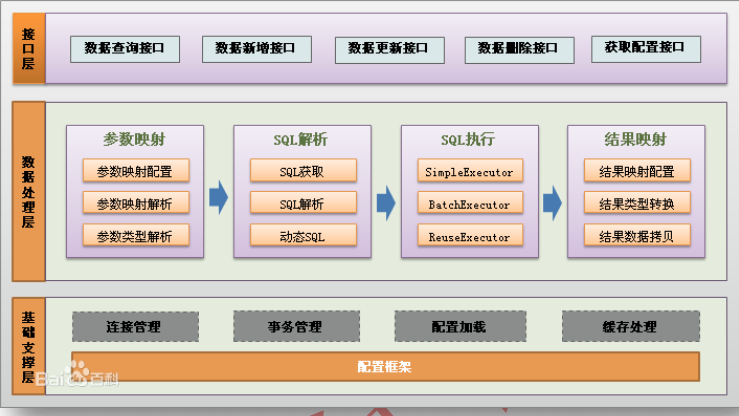
通过xml配置或者注解方式，配置statement

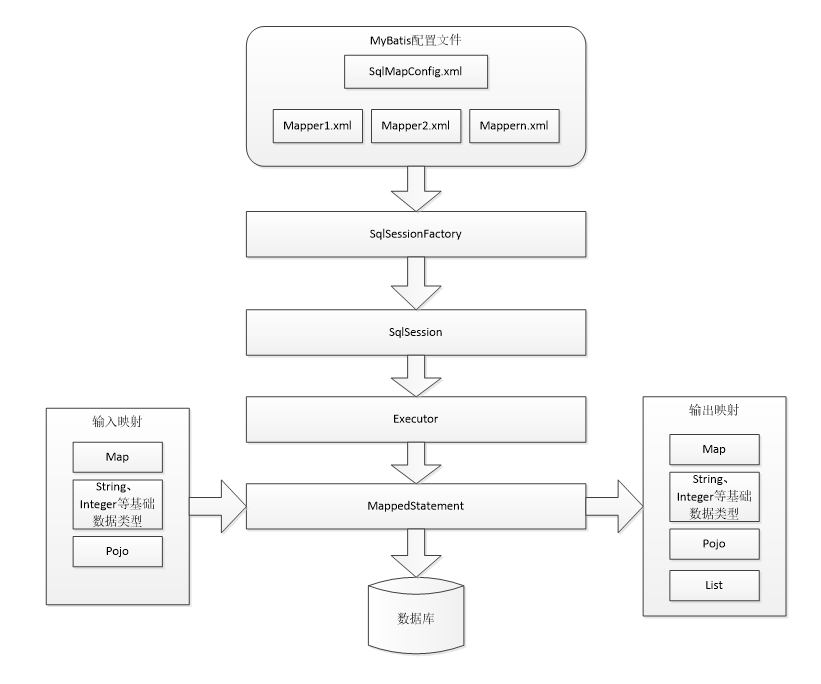
-在xml中配置SQL语句，和代码分离

-需要程序员手动编写SQL语句，方便优化

-简单，上手快

# 体系架构





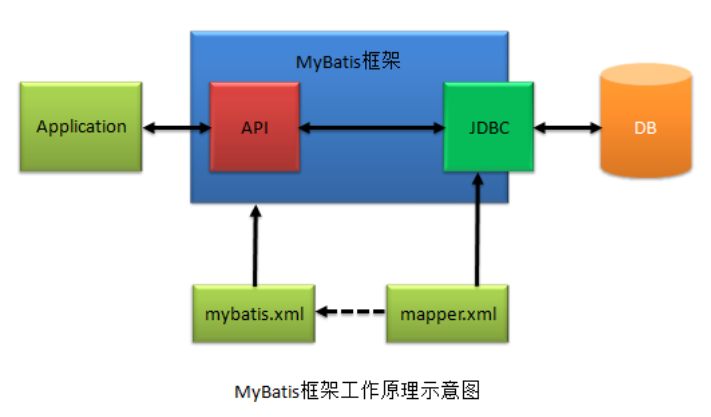
1. mybatis配置

SqlMapConfig.xml，此文件作为mybatis的全局配置文件，配置了mybatis的运行环境等信息。

mapper.xml文件即sql映射文件，文件中配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。

1. 通过mybatis环境等配置信息构造SqlSessionFactory即会话工厂
2. 由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。
3. mybatis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。
4. Mapped Statement也是mybatis一个底层封装对象，它包装了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。
5. Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。
6. Mapped Statement对sql执行输出结果进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。

# 工作原理



# 入门程序

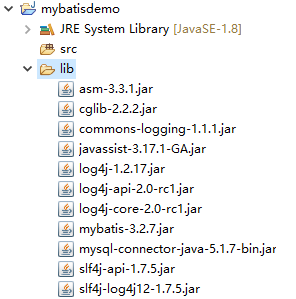
## primary

导包

mybaits包

依赖包（lib目录下）

数据库驱动包



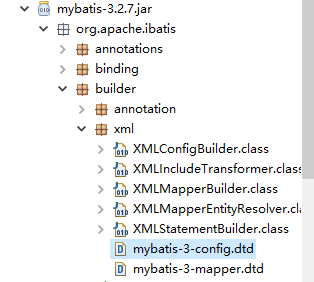
定义实体类

创建空表

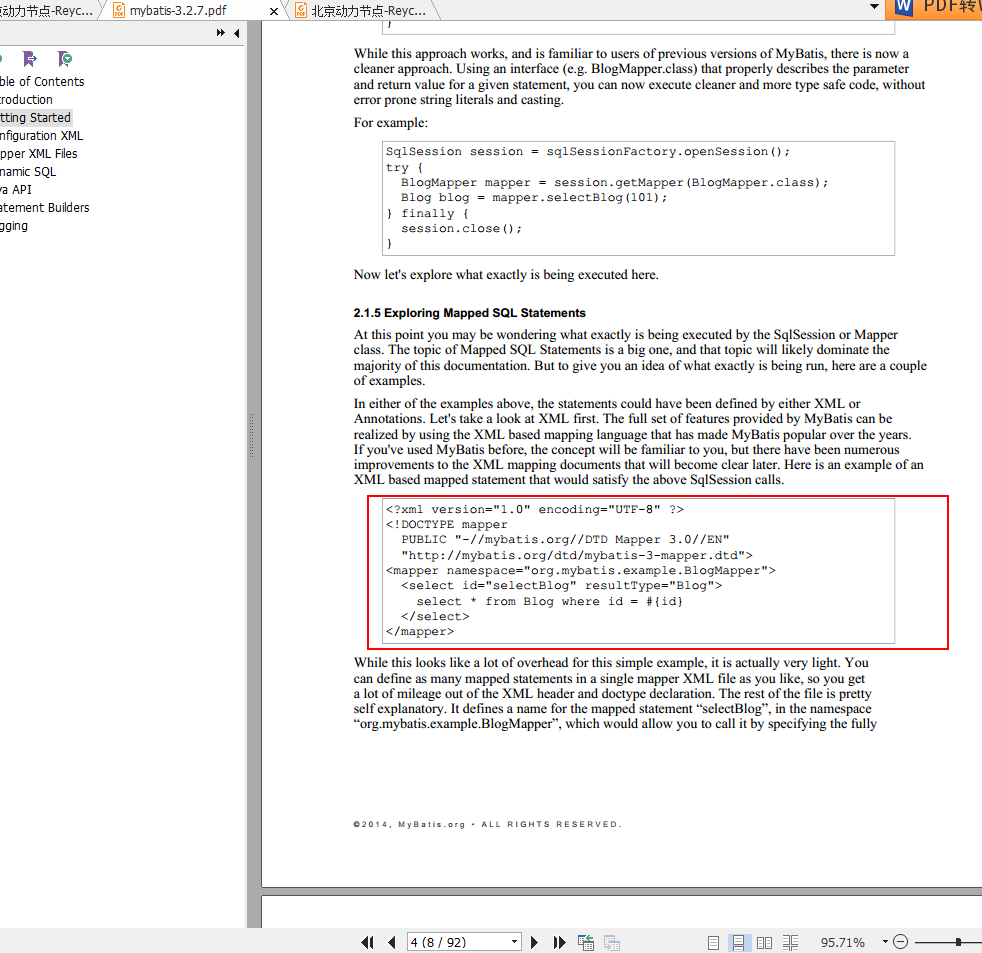
-注意数据库表，字段名和属性名一致

定义Dao接口

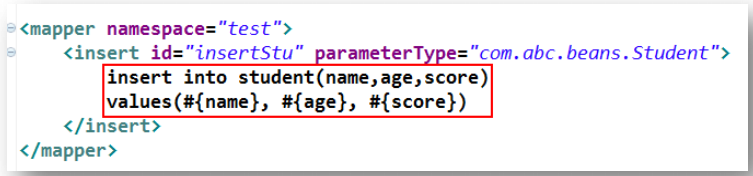
定义映射文件：一般放在dao包中，mapper.xml，约束是mybatis-3-mapper.dtd，存放在org.apache.ibatis.builder.xml中



这个就是文件头了

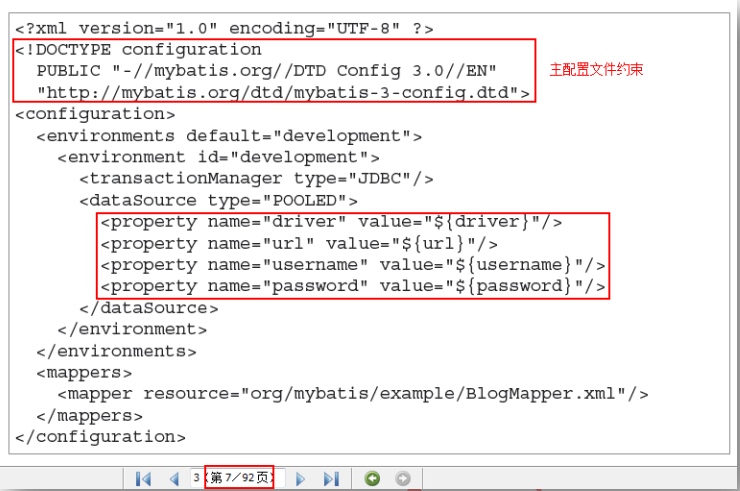


映射文件内容

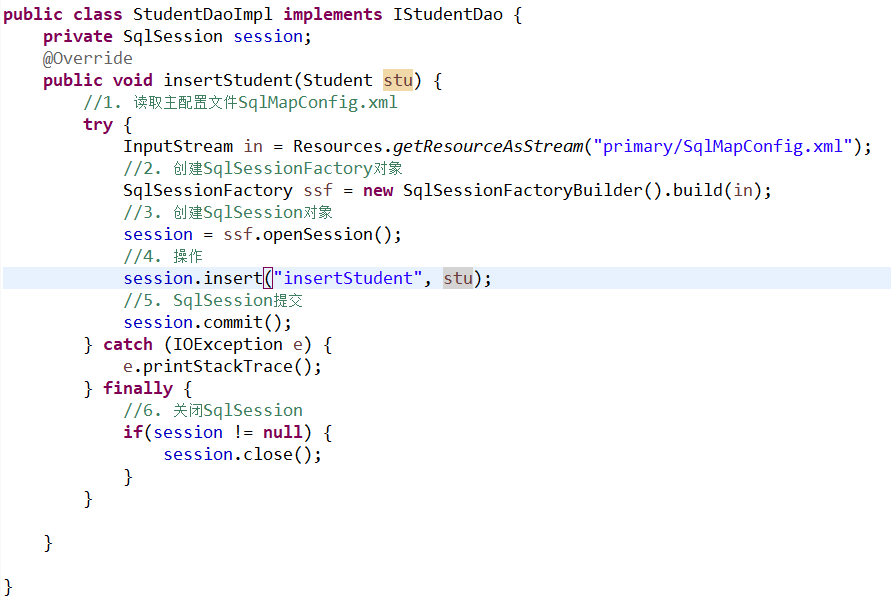


定义主配置文件SqlMapConfig.xml

官方文档中的位置

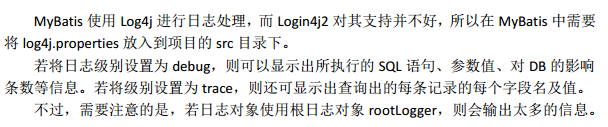


定义dao实现类



定义测试类

添加日志控制文件





最终内容：

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

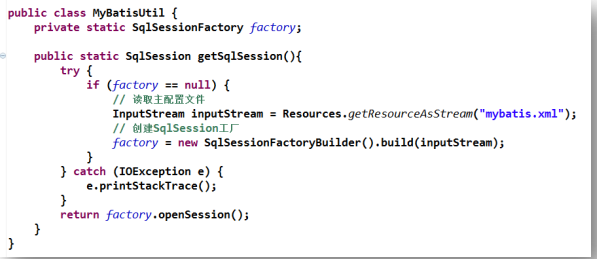
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

# 使用工具类

## primary\_2



创建工具类



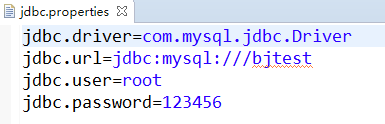
修改实现类

测试通过

# 从jdbc.properties读取数据库连接信息以及多个Mapper

## primary\_3

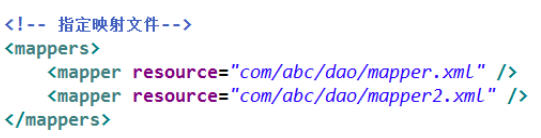
配置文件



修改主配置文件



多个mapper文件注册（mapper里面的namespace+sql\_id要注意）



## 主配置文件

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<properties resource=*"jdbc.properties"*/>

<!-- 别名 包以其子包下所有类 头字母大小都行-->

<typeAliases>

<!-- <typeAlias type="com.itheima.mybatis.pojo.User" alias="User"/> -->

<package name=*"com.itheima.mybatis.pojo"*/>

</typeAliases>

<!-- 和spring整合后 environments配置将废除 -->

<environments default=*"development"*>

<environment id=*"development"*>

<!-- 使用jdbc事务管理 -->

<transactionManager type=*"JDBC"* />

<!-- 数据库连接池 -->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${jdbc.driver}"* />

<property name=*"url"*

value=*"jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8"* />

<property name=*"username"* value=*"root"* />

<property name=*"password"* value=*"root"* />

</dataSource>

</environment>

</environments>

<!-- Mapper的位置 Mapper.xml 写Sql语句的文件的位置 -->

<mappers>

<!-- <mapper resource="sqlmap/User.xml" class="" url=""/> -->

<!-- <mapper resource="sqlmap/User.xml" class="" url=""/> -->

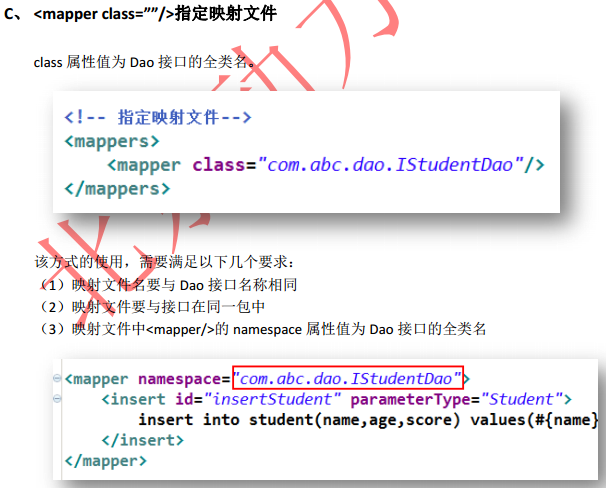
<!-- <mapper class="com.itheima.mybatis.mapper.UserMapper" /> -->

<!-- <mapper url="" /> -->

<package name=*"com.itheima.mybatis.mapper"*/>

</mappers>

</configuration>





## API分析

package primary.dao;

import java.io.IOException;

import java.io.InputStream;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import primary.Student;

public class StudentDaoImpl implements IStudentDao {

private SqlSession session;

@Override

public void insertStudent(Student stu) {

//1. 读取主配置文件SqlMapConfig.xml

try {

InputStream in = Resources.getResourceAsStream("primary/SqlMapConfig.xml");

//2. 创建SqlSessionFactory对象

/\*

\* ssf线程安全

\* SqlSessionFactoryBuilder在创建ssf之后就被销毁，作为方法内局部对象

\* build方法创建好ssf之后，会自动关闭in流

\*/

SqlSessionFactory ssf = new SqlSessionFactoryBuilder().build(in);

//3. 创建SqlSession对象

/\*

\* openSession(true) 自动提交

\* 默认为false，做了以下几件事情

\* 1. 加载主配置文件

\* 2. 创建执行器对象（用于执行映射文件中的SQL语句）

\* 3. 初始化标志变量dirty为false表明数据库未修改

\* 4. 关闭事务自动提交功能

\*

\*/

session = ssf.openSession();

//4. 操作

/\*

\* 常用方法

\* ===底层调用的是update(statement, parameter)方法===

\* insert

\* delete

\* update

\* ==============================================

\* select

\* selectList

\* selectMap

\* selectOne

\* commit

\* rollback

\*

\*/

session.insert("insertStudent", stu);

//5. SqlSession提交

session.commit();

} catch (IOException e) {

e.printStackTrace();

} finally {

//6. 关闭SqlSession

if(session != null) {

session.close();

}

}

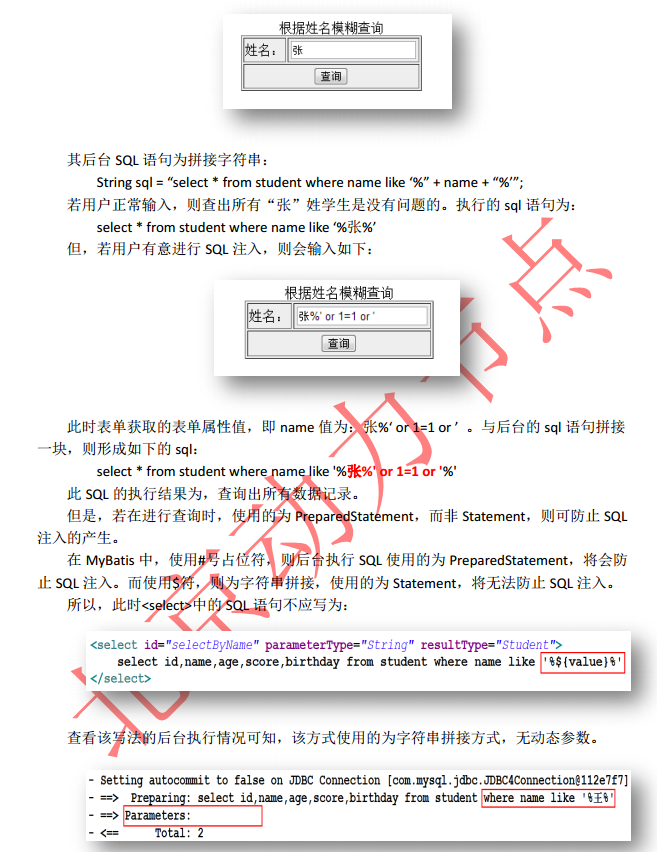
}

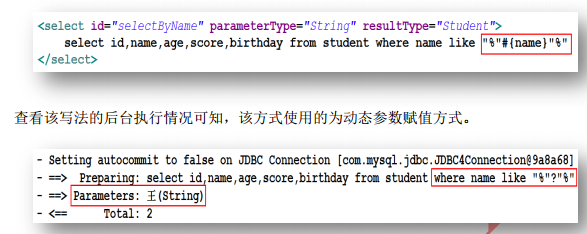
}

# 单表的CRUD操作

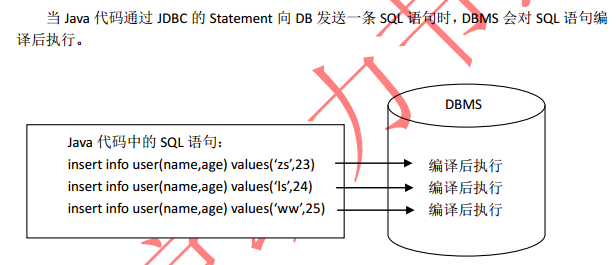
## crud包

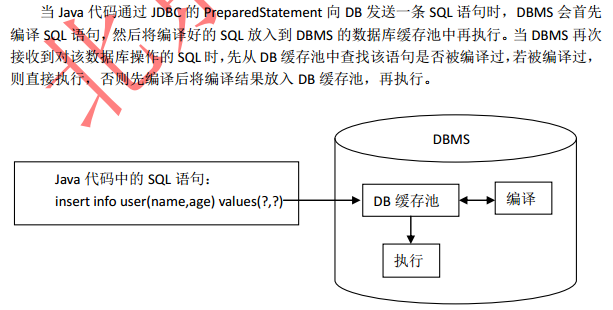
SQL注入问题





SQL的预编译问题





### 属性名和查询字段不相同的处理方法

见mapper.xml的配置

### 动态代理

见TestDemo2

### 多查询条件无法整体接收问题的解决

见IStudentDao.java

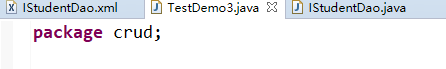
TestDemo2.java

IStudentDao.xml

## 动态SQL

标签

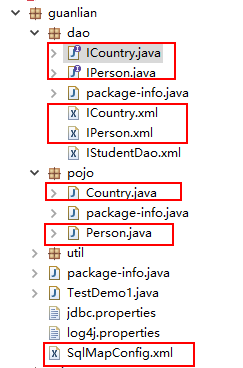
见这3个文件



# 关联查询

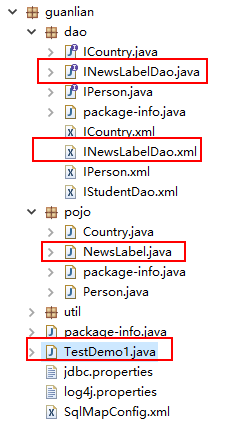
## 多表一对多和多对一

见



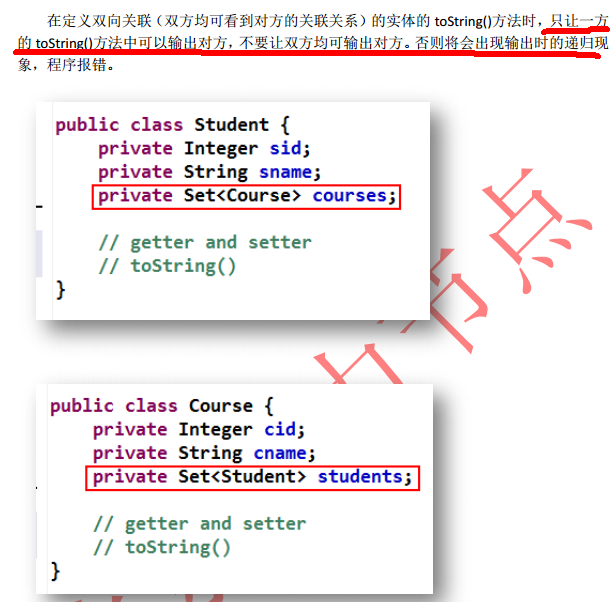
## 自关联查询

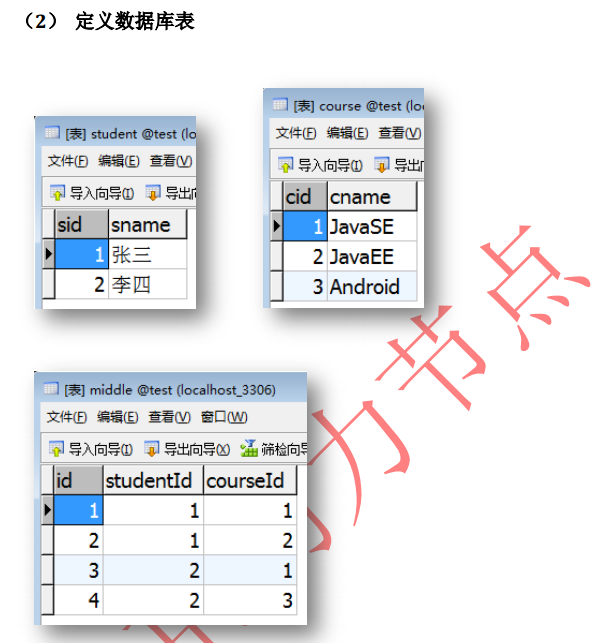
见



## 多表 多对多

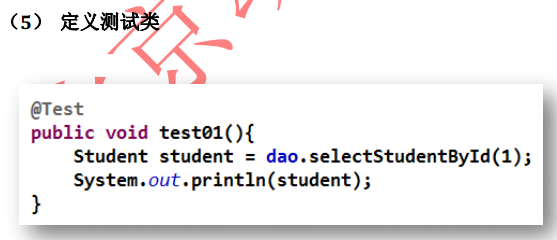
了解







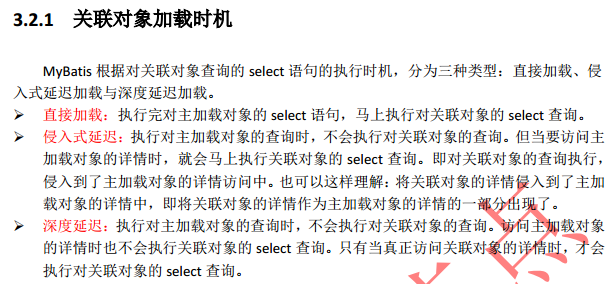


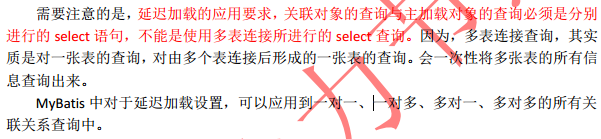


## 延迟加载

减少数据库压力







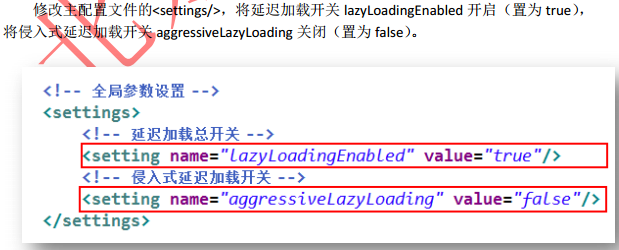
配置：

直接加载

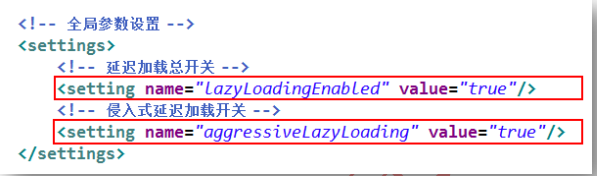


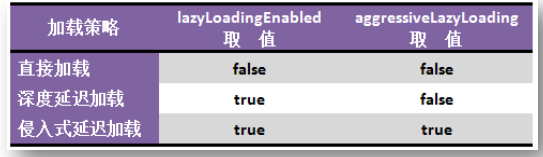
false是直接加载

深度延迟加载：



侵入式延迟加载





# 查询缓存

提高查询访问速度

（了解）

# MyBatis注解式开发

了解