**MyBatis源码阅读准备**

<https://www.jianshu.com/p/e739afb8fe31>

# 导读：阅读源码是提升自己代码能力的一个非常重要的手段，但是源码阅读有时候也非常麻烦，一定要找到入口点，然后动态地去看代码，一步步调试然后画图做笔记记录，才能在一团乱中清理出一个道路来。这篇文章介绍了我在阅读Mybatis源码的准备过程。

**Mybatis简介**   
mybatis是一个流行的半自动映射框架，之所以称为半自动框架是因为它需要手工匹配Java类，SQL和映射关系，而全表映射的Hibernate只需要提供POJO和映射关系就可以了。区别如下表：

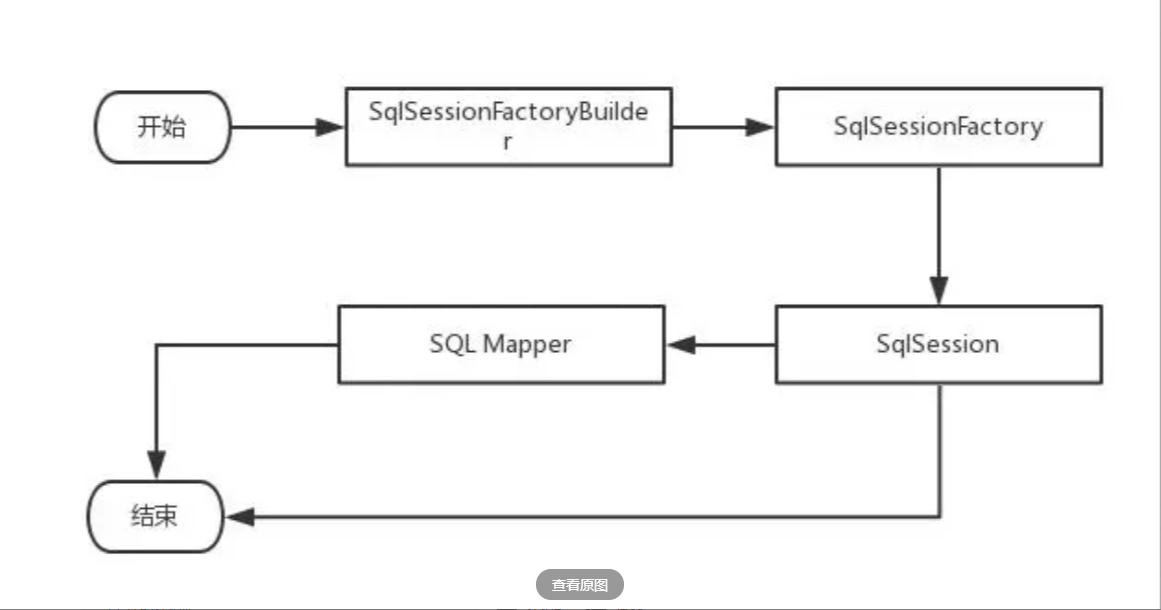
| **MyBatis** | **Hibernate** |
| --- | --- |
| java类 | java类 |
| 映射规则 | 映射规则 |
| SQL | 无 |
|  |  |

通过上表也可以看出，Hibenate无需编写SQL，所以开发效率优于Mybatis.此外，它提供缓存、日志、级联等功能。但是缺点也是十分明显，多表关联复杂的SQL，根据条件变化的SQL。存储过程等场景使用Hibenate非常不便，而性能又难以通过SQL优化。所以注定Hibernate只适用于场景不太复杂，要求性能不太苛刻的时候使用。如果你需要一个灵活的、可以动态生成映射关系的框架，那么MyBatis是一个最好的选择。

**MyBatis组件以及执行基本流程**

* SqlSessionFactoryBuilder（构造器）:它会根据配置信息或者代码生成SqlSessionFactory（工厂接口）
* SqlSessionFactory：依靠工厂来生成SqlSession。
* SqlSession：是一个既可以发送SQL去执行并返回结果的，也可以获取Mapper接口，通过Mapper接口查询并封装数据。
* SQL Mapper:它是MyBatis新设计的组件，它是由一个Java接口和XML文件（或者注解)构成的，需要给出对应的SQL和映射规则。它负责发送SQL去执行，并返回结果。

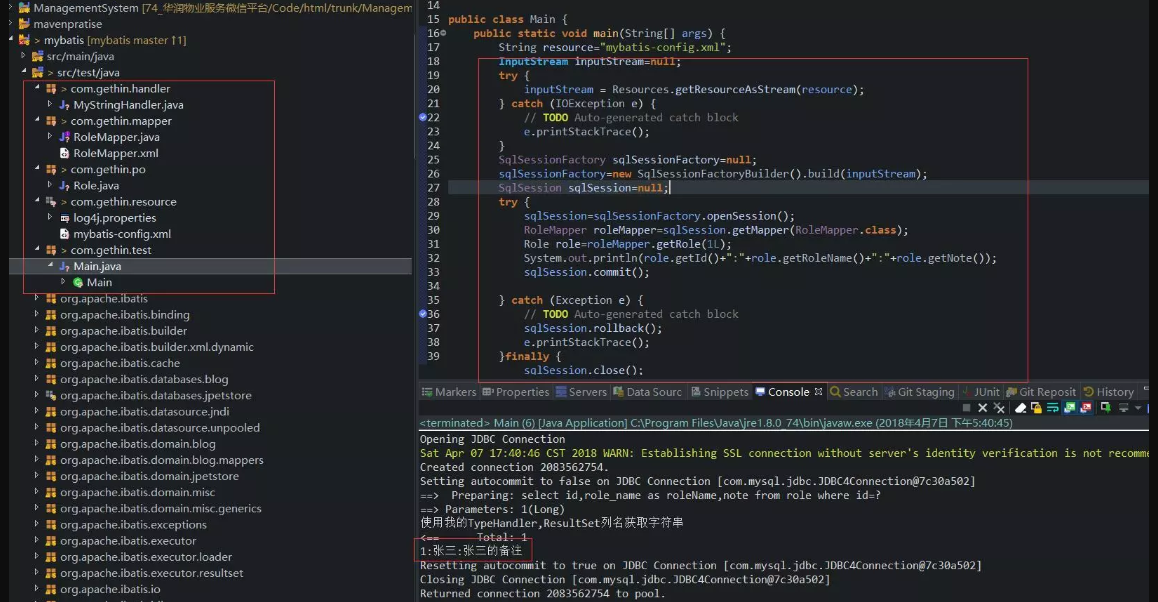
用下图表达上述组件之间的关联



Mybatis 的构成

**mybatis源码阅读准备**

* github上有一个中文注释版的MyBatis源码。[MyBatis中文链接](https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Ftuguangquan%2Fmybatis.git" \t "_blank) <https://github.com/tuguangquan/mybatis>
* Mybatis是以maven管理的，依赖于Mybatis-parent模块。**将mybatis导入到eclipse中时候，也需要将mybatis-parent模块导入。**[mybatis-parent模块链接](https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fmybatis%2Fparent.git" \t "_blank) <https://github.com/mybatis/parent>
* 将模块mybatis，mybatis-parent模块导入到Eclipse中，接下来就通过mybatis的最基本的代码来调试并阅读mybatis源码。整个结构如下图。



image

通过《深入浅出MyBatis技术原理与实战》一书的代码来实现基本调试。代码的目录请看上图，代码的内容下面一一列出。

* 数据库表role

-- ----------------------------

-- Table structure for role

-- ----------------------------

DROP TABLE IF EXISTS `role`;

CREATE TABLE `role` (

`id` int(11) DEFAULT NULL,

`role\_name` varchar(255) DEFAULT NULL,

`note` varchar(255) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of role

-- ----------------------------

INSERT INTO `role` VALUES ('1', '张三', '张三的备注');

* 数据库表role对应的java类

/\*

\* @author gethin

\* 角色的实体类

\*/

public class Role {

private long id;

private String roleName;

private String note;

...

getter setter省略

}

* java RoleMapper接口

/\*

\* @author gethin

\*/

public interface RoleMapper {

public Role getRole(Long id);

public Role findRole(String roleName);

public int deleteRole(Long id);

public int insertRole(Role role);

}

* RoleMapper

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

<!DOCTYPE mapper

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

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

<mapper namespace="com.gethin.mapper.RoleMapper">

<resultMap type="role" id="roleMap">

<id column="id" property="id" javaType="long" jdbcType="BIGINT" />

<result column="role\_name" property="roleName" javaType="string"

jdbcType="VARCHAR" />

<result column="note" property="note"

typeHandler="com.gethin.handler.MyStringHandler" />

</resultMap>

<select id="getRole" parameterType="long" resultMap="roleMap">

select

id,role\_name as roleName,note from role where id=#{id}

</select>

<select id="findRole" parameterType="long" resultMap="roleMap">

select

id,role\_name,note from role where role\_name like CONCAT('%',#{roleName

javaType=string,

jdbcType=VARCHAR,typeHandler=com.gethin.handler.MyStringHandler},'%')

</select>

<insert id="insertRole" parameterType="role">

insert into

role(role\_name,note) value(#{roleName},#{note})

</insert>

<delete id="deleteRole" parameterType="long">

delete from role where

id=#{id}

</delete>

</mapper>

* MyStringHandler

@MappedTypes({String.class})

@MappedJdbcTypes(JdbcType.VARCHAR)

public class MyStringHandler implements TypeHandler<String> {

private Logger log=Logger.getLogger(MyStringHandler.class);

@Override

public String getResult(ResultSet rs, String colName) throws SQLException {

log.info("使用我的TypeHandler,ResultSet列名获取字符串");

return rs.getString(colName);

}

@Override

public String getResult(ResultSet rs, int index) throws SQLException {

log.info("使用我的TypeHandler,ResultSet下标获取字符串");

return rs.getString(index);

}

@Override

public String getResult(CallableStatement cs, int index) throws SQLException {

log.info("使用我的TypeHandler,CallableStatement下标获取字符串");

return cs.getString(index);

}

@Override

public void setParameter(PreparedStatement ps, int index, String value, JdbcType arg3) throws SQLException {

log.info("使用我的TypeHandler");

ps.setString(index, value);

}

}

* log4j.properties 配置文件

log4j.rootLogger=DEBUG,stdout

log4j.logger.org.mybatis=DUBUG

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

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

log4j.appender.logDailyFile.layout.ConversionPattern = %5p %d %C:%m%n

* mybatis-config.xml 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>

<settings>

<setting name="lazyLoadingEnabled" value="true"/>

<!-- <setting name="aggressiveLazyLoading" value="false"/> -->

</settings>

<typeAliases>

<typeAlias alias="role" type="com.gethin.po.Role"/>

</typeAliases>

<typeHandlers>

<typeHandler jdbcType="VARCHAR" javaType="string" handler="com.gethin.handler.MyStringHandler"/>

</typeHandlers>

<!-- 定义数据库的信息，默认使用development数据库构建环境 -->

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

<property name="driver" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/mybatis" />

<property name="username" value="dbuser" />

<property name="password" value="123456" />

</dataSource>

</environment>

</environments>

<!-- 定义映射器 -->

<mappers>

<package name="com.gethin.mapper"/>

</mappers>

</configuration>

* 阅读以及debug入口 （Main）

public class Main {

public static void main(String[] args) {

String resource="mybatis-config.xml";

InputStream inputStream=null;

try {

inputStream = Resources.getResourceAsStream(resource);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

SqlSessionFactory sqlSessionFactory=null;

sqlSessionFactory=new SqlSessionFactoryBuilder().build(inputStream);

SqlSession sqlSession=null;

try {

sqlSession=sqlSessionFactory.openSession();

RoleMapper roleMapper=sqlSession.getMapper(RoleMapper.class);

Role role=roleMapper.getRole(1L);

System.out.println(role.getId()+":"+role.getRoleName()+":"+role.getNote());

sqlSession.commit();

} catch (Exception e) {

// TODO Auto-generated catch block

sqlSession.rollback();

e.printStackTrace();

}finally {

sqlSession.close();

}

}

}

**通过执行Main类，并设立断点，就可以一步一步进行源码的调试以及阅读了。**

作者：d3f59bfc7013  
链接：https://www.jianshu.com/p/e739afb8fe31  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。