前三次课总结

第一节课，我们分析了 MyBatis 的核心特性和一些高级用法。

第二节课，我们学习了 Mybatis 的整体架构和模块分层，梳理了 MyBatis 主要的工作流程，并且通过阅读 MyBatis 的源码，理解了它工作原理。

第三节课，我们学习了 MyBatis 的插件机制，包括插件的工作原理和应用场景。另一个就是 Spring 集成 MyBatis 的原理，我们是怎么做到不用手动创建 SqlSession，直接把 Mapper 接口注入到 Service 层就可以使用的。

课程目标

1、 实现 1.0 版本，掌握 MyBatis 的本质、核心功能、核心对象、执行流程

2、 通过分析 2.0 版本，体验框架的演进过程，理解 MyBatis 设计思想与细节

内容定位

适合已经掌握 MyBatis 的基本使用，理解了 MyBatis 的工作原理，想要进一步理解 MyBatis 为什么这么设计的同学。

一、MeBatis 需求分析

假如你在一家软件公司的研发部工作，有一天技术总监老王想让你负责开发一个项目，你要做的第一件事情是什么？

确定需求。

那我们要开发这个项目，需求从哪里来？ 我们要跟老王沟通下。

1、项目目标：为什么要做这个项目？做成什么样？

老王说：我发现在业务复杂的项目中，开发的兄弟们用 JDBC 操作数据库太麻烦了，想要把一些基础的操作做一个封装和提取，让开发的兄弟们更加专注于业务的开发，这样就可以提升开发效率，远离 996。

原来是一个操作数据库的框架。

那么我要问一下老王：这个项目要做什么，才简化我们对数据库的操作呢？或者说，在业务复杂的项目中使用 JDBC 操作数据库，麻烦在哪里？

2、核心功能：这个框架需要解决什么问题？

老王给我看了一段 JDBC 的代码：

```
rs.close();
stmt.close();
conn.close();
```

1）它需要实现对连接资源的自动管理，也就是把创建 Connection、Statement、关闭 Connection、Statement、ResultSet 这些操作封装到底层的对象中，不需要在应用层手动调用。

```
String sql = "SELECT bid, name, author_id FROM blog where bid = 1";
ResultSet rs = stmt.executeQuery(sql);
```

2)它需要把 SQL 语句抽离出来实现集中管理 ,开发人员不用在业务代码里面写 SQL 语句。

```
Integer bid = rs.getInt("bid");
String name = rs.getString("name");
Integer authorId = rs.getInt("author_id");
blog.setAuthorId(authorId);
blog.setBid(bid);
blog.setName(name);
```

3）它需要实现对结果集的转换，也就是我们指定了映射规则之后，这个框架会自动帮我们把 ResultSet 映射成实体类对象。

4）做了这些事以后，这个框架需要提供一个 API 来给我们操作数据库，这里面封装了对数据库的操作的常用的方法。

3、功能分解：这个框架要怎么解决这些问题？

老王的需求我已经了解了，这个框架应该怎么解决这些问题呢？

我们先来分析一下需要哪些核心对象：

1）核心对象

1、存放参数和结果映射关系、存放 SQL 语句，我们需要定义一个配置类；

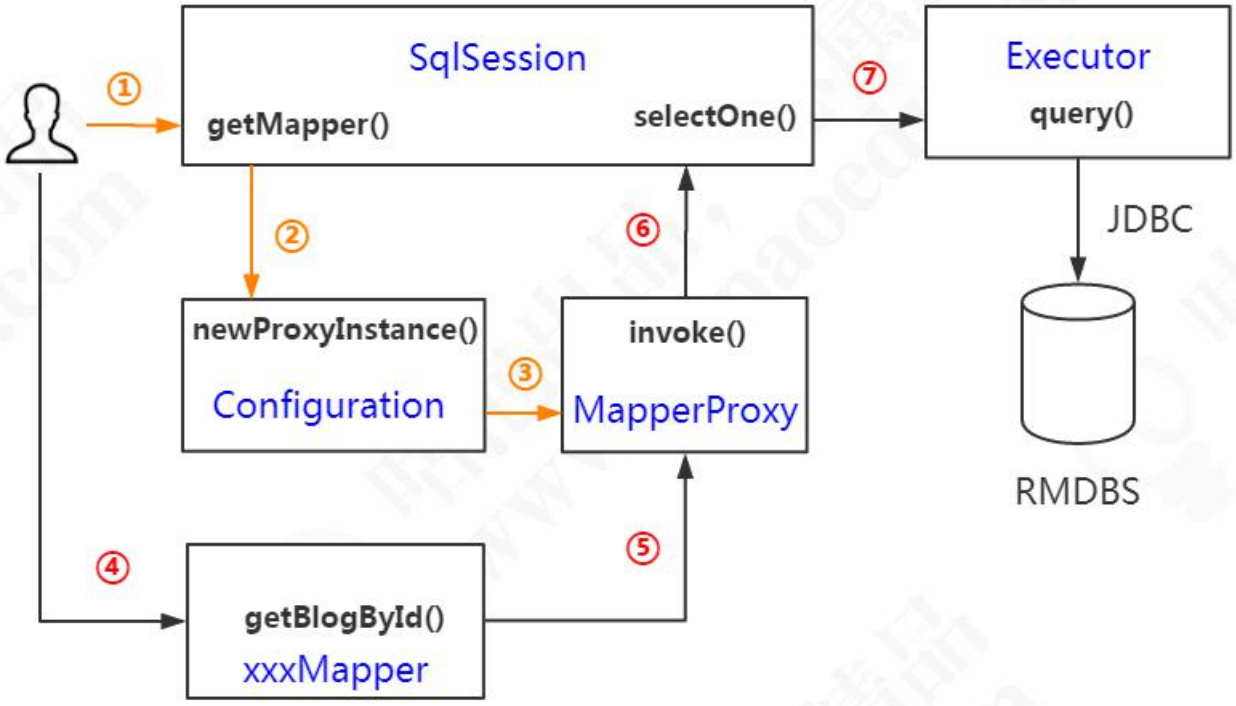
2、执行对数据库的操作，处理参数和结果集的映射，创建和释放资源，我们需要定义一个执行器；

3、有了这个执行器以后，我们不能直接调用它，而是定义一个给应用层使用的 API，它可以根据 SQL 的 id 找到 SQL 语句，交给执行器执行；

4、直接使用 id 查找 SQL 语句太麻烦了，我们干脆把存放 SQL 的命名空间定义成一个接口，把 SQL 的 id 定义成方法，这样只要调用接口方法就可以找到要执行的 SQL。这个时候我们需要引入一个代理类。

核心对象有了，接下来我们分析一下这个框架操作数据库的主要流程，先从单条查询入手。

2）操作流程（绘图）



1、定义接口 Mapper 和方法，用来调用数据库操作。

Mapper 接口操作数据库需要通过代理类。

2、定义配置类对象 Configuration。

3、定义应用层的 API SqlSession。它有一个 getMapper()方法，我们会从配置类 Configuration 里面使用 Proxy.newProxyInatance()拿到一个代理对象 MapperProxy。

4、有了代理对象 MapperProxy 之后，我们调用接口的任意方法，就是调用代理对象的 invoke()方法。

5、代理对象 MapperProxy 的 invoke()方法调用了 SqlSession 的 selectOne()。

6、SqlSession 只是一个 API，还不是真正的 SQL 执行者，所以接下来会调用执行

器 Executor 的 query()方法。

7、执行器 Executor 的 query()方法里面就是对 JDBC 底层的 Statement 的封装，最终实现对数据库的操作，和结果的返回。

基于我们总结的这个框架的主要工作流程，接下来我们就要动手去写这个框架了。

我们先给它起个名字叫 MeBatis。

二、V1.0 的实现

创建一个全新的 maven 工程，命名为 mebatis，引入 mysql 的依赖。

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.21</version>
</dependency>
```

1、SqlSession

我们已经分析了 MeBatis 的主要对象和操作流程，应该从哪里入手？

当我们在 psvm 操作的时候，第一个需要的对象是 SqlSession。所以我们从应用层的接口 SqlSession 入手。

那么我们先来创建一个 package，它是我们手写的 MeBatis ，我们建一个包叫 mebatis。

首先我们创建一个自己的 SqlSession，叫 GPSqlSession。

根据我们刚才总结的流程图，SqlSession 需要有一个获取代理对象的方法，那么这个代理对象是从哪里获取到的呢？是从我们的配置类里面获取到的，因为配置类里面有

接口和它要产生的代理类的对应关系。

所以，我们要先持有一个 Configuration 对象，叫 GPConfiguration，我们也创建这个类。除了获取代理对象之外，Configuration 里面还存储了我们的接口方法（也就是 statementId）和 SQL 语句的绑定关系。

第二个，我们在 SqlSession 中定义的操作数据库的方法，最后都会调用 Executor 去操作数据库，所以我们还要持有一个 Executor 对象，叫 GPExecutor，我们也创建它。

```
// GPSqlSession.java
private GPConfiguration configuration;
private GPExecutor executor;
```

除了这两个属性之外，我们还要定义 SqlSession 的行为，也就是它的主要的方法。

第一个方法是查询方法，selectOne()，由于它可以返回任意类型，我们把返回值定义成<T> T 泛型。selectOne()有两个参数，一个是 String 类型的 statementId，我们会根据它找到 SQL 语句。一个是 Object 类型的 parameter 参数（可以是 Integer 也可以是 String 等等，任意类型），用来填充 SQL 里面的占位符。

它会调用 Executor 的 query()方法，所以我们创建 Executor 类，传入这两个参数，一样返回一个泛型。Executor 里面要传入 SQL，但是我们还没拿到，先用 statementId 代替。

```
// GPSqlSession.java
public <T> T selectOne(String statementId, Object parameter){
    String sql = statementId; // 先用 statementId 代替 SQL
    return executor.query(sql, parameter);
}
```

```
// GPExecutor.java
public <T> T query(String sql, Object paramater ) {
    return null;
}
```

第二个方法是获取代理对象的方法，我们通过这种方式去避免了 statementId 的硬编码。

我们在 SqlSession 中创建一个 getMapper()的方法，由于可以返回任意类型的代理类，所以我们把返回值也定义成泛型<T> T。我们是根据接口类型获取到代理对象的，所以传入参数要用类型 Class。

```
// GPSqlSession.java
public <T> T getMapper(Class clazz){
    return null;
}
```

2、Configuration

代理对象我们不是在 SqlSession 里面获取到的，要进一步调用 Configuration 的 getMapper()方法。返回值需要强转成(T)。

```
// GPSqlSession.java
public <T> T getMapper(Class<T> clazz){
    return (T)configuration.getMapper(clazz);
}
```

我们先在 Configuration 创建这个方法，返回类型一样是泛型<T> T，先返回空。

```
// GPConfiguration.java
public <T> T getMapper(Class clazz) {
    return null;
}
```

3、MapperProxy

我们要在 Configuration 中通过 getMapper()方法拿到这个代理对象，必须要有一个实现了 InvocationHandler 的代理类。我们来创建它：GPMapperProxy。

提供一个 invoke()方法。

```
// GPMapperProxy.java
public class GPMapperProxy implements InvocationHandler {
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        return null;
    }
}
```

invoke()的实现我们先留着，先返回 null。MapperProxy 已经有了，我们回到 Configuration.getMapper()完成获取代理对象的逻辑。

返回代理对象，直接使用 JDK 的动态代理：第一个参数是类加载器，第二个参数是被代理类，第三个参数是代理类。

把返回结果强转为(T)：

```
// GPConfiguration.java
public <T> T getMapper(Class<T> clazz, GPSqlSession sqlSession) {
        return (T)Proxy.newProxyInstance(this.getClass().getClassLoader(),
                new Class[]{clazz},
                new GPMapperProxy());
}
```

获取代理类的逻辑已经实现完了，我们可以在 SqlSession 中通过 getMapper()拿到代理对象了，也就是可以调用 invoke()方法了。接下来去完成 MapperProxy 的 invoke()方法。

在 MapperProxy 的 invoke()方法里面又调用了 SqlSession 的 selectOne()方法。一个问题出现了：在 MapperProxy 里面根本没有 SqlSession 对象？

这两个对象的关系怎么建立起来？MapperProxy 怎么拿到一个 SqlSession 对象？很简单，我们可通过构造函数传入它。

先定义一个属性，然后在 MapperProxy 的构造函数里面赋值：

```
// GPMapperProxy.java
private GPSqlSession sqlSession;
```

```
public GPMapperProxy(GPSqlSession sqlSession) {
    this.sqlSession = sqlSession;
}
```

因为修改了代理类的构造函数，这个时候 Configuration 创建代理类的方法 getMapper()也要修改。

问题：Configuration 的 getMapper()方法参数中也没有 SqlSession，没办法传给 MapperProxy 的构造函数。怎么拿到 SqlSession 呢？是直接 new 一个吗？

不需要，可以在 SqlSession 调用它的时候直接把自己传进来（红色是修改的地方）：

```
// GPConfiguration.java
public <T> T getMapper(Class clazz, GPSqlSession sqlSession) {
    return (T)Proxy.newProxyInstance(this.getClass().getClassLoader(),
            new Class[]{clazz},
            new GPMapperProxy(sqlSession));
}
```

那么 SqlSession 的 getMapper()方法也要修改（红色是修改的地方）：

```
// GPSqlSession.java
public <T> T getMapper(Class clazz){
    return configuration.getMapper(clazz, this);
}
```

现在在 MapperProxy 里面已经就可以拿到 SqlSession 对象了，在 invoke()方法里面我们会调用 SqlSession 的 selectOne()方法。我们继续来完成 invoke()方法。

selectOne()方法有两个参数， statementId 和 paramater，这两个我们怎么拿到呢？

statementId 其实就是接口的全路径+方法名，中间加一个英文的点。

paramater 可以从方法参数中拿到，这里我们只传了一个参数，用 args[0]。

它要把 statementId 和参数传给 SqlSession：

```
// GPMapperProxy.java
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    String mapperInterface = method.getDeclaringClass().getName();
```

```
    String methodName = method.getName();
    String statementId = mapperInterface + "." + methodName;

    return sqlSession.selectOne(statementId, args[0]);
}
```

4、Executor

到了 sqlSession 的 selectOne()方法 ,这里我们要去调用 Executor 的 query()方法，这个时候我们必须传入 SQL 语句和参数（根据 statementId 获取）。

问题来了 :我们怎么根据 StatementId 找到我们要执行的 SQL 语句呢？他们之间的绑定关系我们配置在哪里？

为了简便，免去读取文件流和解析 XML 标签的麻烦，我们把我们的 SQL 语句放在 Properties 文件里面。

我们在 resources 目录下创建一个 mesql.properties 文件。key 就是接口全路径+方法名称，SQL 是我们的查询 SQL。

参数这里，因为我们要传入一个整数，所以先用一个%d 的占位符代替：

（这里直接把 standalone 工程的实体类 Blog 和 BlogMapper 接口复制过来）

```
com.gupaoedu.mebatis.BlogMapper.selectBlogById=select * from blog where bid = %d
```

这个绑定关系是放在配置类 Configuration 里面的。

为了避免重复解析，我们在 Configuration 创建一个静态属性和静态方法，直接解析 mesql.properties 文件里面的所有 KV 键值对：

```
// GPConfiguration.java
public static final ResourceBundle sqlMappings;

static{
    sqlMappings = ResourceBundle.getBundle("mesql");
}
```

这样就可以通过 Configuration 拿到 SQL 了。

如果 SQL 语句拿不到，说明不存在映射关系（或者不是接口中定义的操作数据的方法，比如 toString()），我们返回空。

```
// GPSqlSession.java
public <T> T selectOne(String statement, String parameter){
    String sql = GPConfiguration.sqlMappings.getString(statement);
    if( null != sql && !"".equals(sql)){
        return executor.query(sql, parameter);
    }
    return null;
}
```

SQL 语句已经拿到了，接下来就是 Executor 类的 query()方法，Executor 是数据库操作的真正执行者。它里面应该做什么事情？

我们干脆直接把 JDBC 的代码全部复制过来，职责先不用细分。

参数用传入的参数替换%d 占位符，需要 format 一下。

```
// GPExecutor.java
ResultSet rs = stmt.executeQuery(String.format(sql, paramater));
```

最后我们把结果强转一下。

```
// GPExecutor.java
return (T)blog;
```

写一个测试类：

```
// MeBatisTest.java
public class MeBatisTest {
    public static void main(String[] args) {
        GPSqlSession sqlSession = new GPSqlSession();
        BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
        blogMapper.selectBlogById(1);
```

```
    }
}
```

跑一下：

```
Exception in thread "main" java.lang.NullPointerException
        at com.gupaoedu.hand.GPSqlSession.getMapper(GPSqlSession.java:23)
    at com.gupaoedu.hand.MyBatisBoot.main(MyBatisBoot.java:13)
```

configuration 是空的，忘记拿到 Configuration 了！那么 Executor 肯定也是空的。

构造函数里面要给他们俩加上：

```
// GPSqlSession.java
public GPSqlSession(GPConfiguration configuration, GPExecutor executor){
    this.configuration = configuration;
    this.executor = executor;
}
```

改一下我们的测试类（红色是修改部分）：

```
// MeBatisTest.java
public class MeBatisTest {
    public static void main(String[] args) {
        GPSqlSession sqlSession = new GPSqlSession(new GPConfiguration(), new GPExecutor());
        BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
        blogMapper.selectBlogById(1);
    }
}
```

测试通过，MeBatis 1.0 的版本完成了：

```
Blog{bid=1, name='MyBatis 源码分析', authorId='1001'}
```

三、1.0 的不足

MeBatis1.0 的功能完成了，在拿给老王看之前，我抽了根烟思考了一下：

V1.0 的不足

1、在 Executor 中，对参数、语句和结果集的处理是耦合的，没有实现职责分离；

2、参数：没有实现对语句的预编译，只有简单的格式化（format），效率不高，还存在 SQL 注入的风险；

3、语句执行：数据库连接硬编码；

4、结果集：还只能处理 Blog 类型，没有实现根据实体类自动映射。

确实有点搓，拿不出手。

V1.0 的优化目标

支持参数预编译；

支持结果集的自动处理（通过反射）；

对 Executor 的职责进行细化。

V1.0 的功能增强目标

在方法上使用注解配置 SQL；

查询带缓存功能；

支持自定义插件。

四、V2.0 的实现

源码工程：gupaoedu-vip-mybatis-custom

1、配置文件

创建了全局配置文件 mybatis.properties，存放 SQL 连接信息、缓存开关、插件地址、Mapper 接口地址。

全局配置文件在 Configuration 配置类的构造器中解析。

2、参数处理

创建 ParameterHandler ,调用 psmt 的 set 方法。propertie 文件中 SQL 语句的%d 占位符改成？。

3、结果集处理

创建 ResultSetHandler，在其中创建 pojo 对象，获取 ResultSet 值，通过反射给 pojo 对象赋值。

实体类的转换关系通过@Entity 注解（保存在 MapperRegistry 中），从 MapperProxyFactory（构造函数）——MapperProxy 一路传递到 ResultSetHandler 中。

4、语句执行处理

创建 StatementHandler，在 Executor 中调用。封装获取连接的方法。

执行查询前调用 ParameterHandler，执行查询后调用 ResultSetHandler。

5、支持注解配置 SQL

定义了一个@Select 注解，加在方法上。

在 Configuration 构造函数中的 parsingClass() 中解析，保存在

mappedStatements 中（一个 HashMap）。

注意：在 properties 中和注解上同时配置 SQL 语句，注解会覆盖 properties。

properties 中对表达三个对象的映射关系并不适合，所以暂时用--分隔。注意类型前面不能有空格。

6、支持查询缓存

定义了一个 CachingExecutor，当全局配置中的 cacheEnabled=true 时，Configuration 的 newExecutor()方法会对 SimpleExecutor 进行装饰，返回被装饰过的 Executor。CachingExecutor 中用 HashMap 维护缓存。

在 DefaultSqlSession 调用 Executor 时，会先走到装饰器 CachingExecutor。

定义了一个 CacheKey 用于计算缓存 Key，主要根据 SQL 语句和参数计算。

7、支持插件

定义了一个@Intercepts 注解，目前还只能拦截 Executor 的方法，所以属性只要配置方法名称。

定义 Interceptor 接口，是所有自定义插件必须实现的接口。

定义 InterceptorChain 容器，用来存放解析过的拦截器。在 Configuration 中创建 Executor 的时候，会调用它的 pluginAll()方法，对 Executor 循环代理。

定义 Invocation 包装类，用于在执行完自定义插件逻辑后调用 Executor 的原方法。

定义 Plugin 代理类，提供了一个 wrap()方法用于产生代理对象。当 Executor 被代理后，所有的方法都会走到 invoke()方法中，进一步调用自定义插件的 intercept()方法。

完成了这些功能，我觉得应该可以拿给老王看了。

五、V2.0 可优化之处

老王看了 mebatis 2.0 的代码以后，点了一根烟，提了一些建议：

1、在 ResultSetHandler 中，类型处理都是写死的，能不能创建一个 TypeHandler，把这些关系维护起来，处理所有类型的转换关系和自定义类型；

2、只实现了@Select 的注解 ,插入、删除、修改的注解呢？参数能不能用@Param 传入类型？

3、插件只能拦截 Executor，能不能实现对其他核心对象的方法的拦截？插件可以支持配置参数么？

4、缓存只有一级，不能在单个方法上关闭（properties 不够用了），能不能实现多级的缓存？

5、异常处理有点粗暴，都是直接 catch，没有细化；

……

小哥，接下来拯救世界的任务就交给你了……

作业汇总

1、resultType 和 resultMap 的区别？

resultType 是<select>标签的一个属性，适合简单对象（POJO、JDK 自带类型：Integer、String、Map 等），只能自动映射，适合单表简单查询。

```
<select id="selectAuthor" parameterType="int" resultType="com.gupaoedu.domain.Author">
    select author_id authorId, author_name authorName
    from author where author_id = #{authorId}
</select>
```

resultMap 是一个可以被引用的标签，适合复杂对象，可指定映射关系，适合关联复合查询。

```
<resultMap id="BlogWithAuthorResultMap"
type="com.gupaoedu.domain.associate.BlogAndAuthor">
    <id column="bid" property="bid" jdbcType="INTEGER"/>
    <result column="name" property="name" jdbcType="VARCHAR"/>
    <!-- 联合查询，将 author 的属性映射到 ResultMap -->
    <association property="author" javaType="com.gupaoedu.domain.Author">
        <id column="author_id" property="authorId"/>
        <result column="author_name" property="authorName"/>
    </association>
</resultMap>
```

2、collection 和 association 的区别？

association：一对一

```
<!-- 另一种联合查询(一对一)的实现，但是这种方式有“N+1”的问题 -->
<resultMap id="BlogWithAuthorQueryMap" type="com.gupaoedu.domain.associate.BlogAndAuthor">
    <id column="bid" property="bid" jdbcType="INTEGER"/>
    <result column="name" property="name" jdbcType="VARCHAR"/>
    <association property="author" javaType="com.gupaoedu.domain.Author"
                 column="author_id" select="selectAuthor"/> <!-- selectAuthor 定义在下面
-->
</resultMap>
```

collection：一对多、多对多

```
<!--  查询文章带评论的结果（一对多） -->
<resultMap id="BlogWithCommentMap" type="com.gupaoedu.domain.associate.BlogAndComment"
```

```
extends="BaseResultMap" >
    <collection property="comment" ofType="com.gupaoedu.domain.Comment">
        <id column="comment_id" property="commentId" />
        <result column="content" property="content" />
    </collection>
</resultMap>
```

```
<!-- 按作者查询文章评论的结果（多对多） -->
<resultMap id="AuthorWithBlogMap" type="com.gupaoedu.domain.associate.AuthorAndBlog" >
    <id column="author_id" property="authorId" jdbcType="INTEGER"/>
    <result column="author_name" property="authorName" jdbcType="VARCHAR"/>
    <collection property="blog" ofType="com.gupaoedu.domain.associate.BlogAndComment">
        <id column="bid" property="bid" />
        <result column="name" property="name" />
        <result column="author_id" property="authorId" />
        <collection property="comment" ofType="com.gupaoedu.domain.Comment">
            <id column="comment_id" property="commentId" />
            <result column="content" property="content" />
        </collection>
    </collection>
</resultMap>
```

3、PrepareStatement 和 Statement 的区别？

两个都是接口，PrepareStatement 是继承自 Statement 的；

Statement 处理静态 SQL，PreparedStatement 主要用于执行带参数的语句；

PreparedStatement 的 addBatch()方法一次性发送多个查询给数据库；

PS 相似 SQL 只编译一次（对语句进行了缓存，相当于一个函数），减少编译次数；

PS 可以防止 SQL 注入；

MyBatis 默认值：PREPARED

4、跟踪 update()流程，绘制每一步的时序图（4 个）

自行绘制。

5、总结：MyBatis 里面用到了哪些设计模式？（已讲解）

第三次课已讲解，笔记中有。

6、当我们传入 RowBounds 做翻页查询的时候，使用 limit 物理分页，代替原来的逻辑分页

基于 mybatis-standalone，MyBatisTest.java —— testSelectByRowBounds()

>代码在 interceptor 包中

7、在未启用日志组件的情况下，输出执行的 SQL，并且统计 SQL 的执行时间（先实现查询的拦截）

>代码在 interceptor 包中

面试题总结

1、MyBatis 解决了什么问题？

或：为什么要用 MyBatis？

或：MyBatis 的核心特性？

1）资源管理（底层对象封装和支持数据源）

2）结果集自动映射

3）SQL 与代码分离，集中管理

4）参数映射和动态 SQL

5）其他：缓存、插件等

2、MyBatis 编程式开发中的核心对象及其作用？

SqlSessionFactoryBuilder 创建工厂类

SqlSessionFactory 创建会话

SqlSession 提供操作接口

MapperProxy 代理 Mapper 接口后，用于找到 SQL 执行

3、Java 类型和数据库类型怎么实现相互映射？

通过 TypeHandler，例如 Java 类型中的 String 要保存成 varchar，就会自动调用相应的 Handler。如果没有系统自带的 TypeHandler，也可以自定义。

4、SIMPLE/REUSE/BATCH 三种执行器的区别？

SimpleExecutor 使用后直接关闭 Statement：closeStatement(stmt);

```
//SimpleExecutor.java
public int doUpdate(MappedStatement ms, Object parameter) throws SQLException {
  Statement stmt = null;
  //中间省略……
  } finally {
    closeStatement(stmt);
  }
}
```

ReuseExecutor 放在缓存中，可复用：PrepareStatement——getStatement()

```
//ReuseExecutor.Java
public int doUpdate(MappedStatement ms, Object parameter) throws SQLException {
  //中间省略……
  Statement stmt = prepareStatement(handler, ms.getStatementLog());
  //中间省略……
}

private Statement prepareStatement(StatementHandler handler, Log statementLog) throws
SQLException {
  Statement stmt;
  //中间省略……
  if (hasStatementFor(sql)) {
    stmt = getStatement(sql);
  //中间省略……
}

private Statement getStatement(String s) {
  return statementMap.get(s);
}
```

BatchExecutor 支持复用且可以批量执行 update()，通过 ps.addBatch()实现 handler.batch(stmt);

```
//BatchExecutor.Java
public int doUpdate(MappedStatement ms, Object parameterObject) throws SQLException {
  //中间省略……
  final Statement stmt;
  //中间省略……
    stmt = statementList.get(last);
    //中间省略……
    statementList.add(stmt);
    batchResultList.add(new BatchResult(ms, sql, parameterObject));
  }
  handler.batch(stmt);
}
```

7、MyBatis 一级缓存与二级缓存的区别？

一级缓存 :在同一个会话(SqlSession)中共享 ,默认开启 ,维护在 BaseExecutor 中。

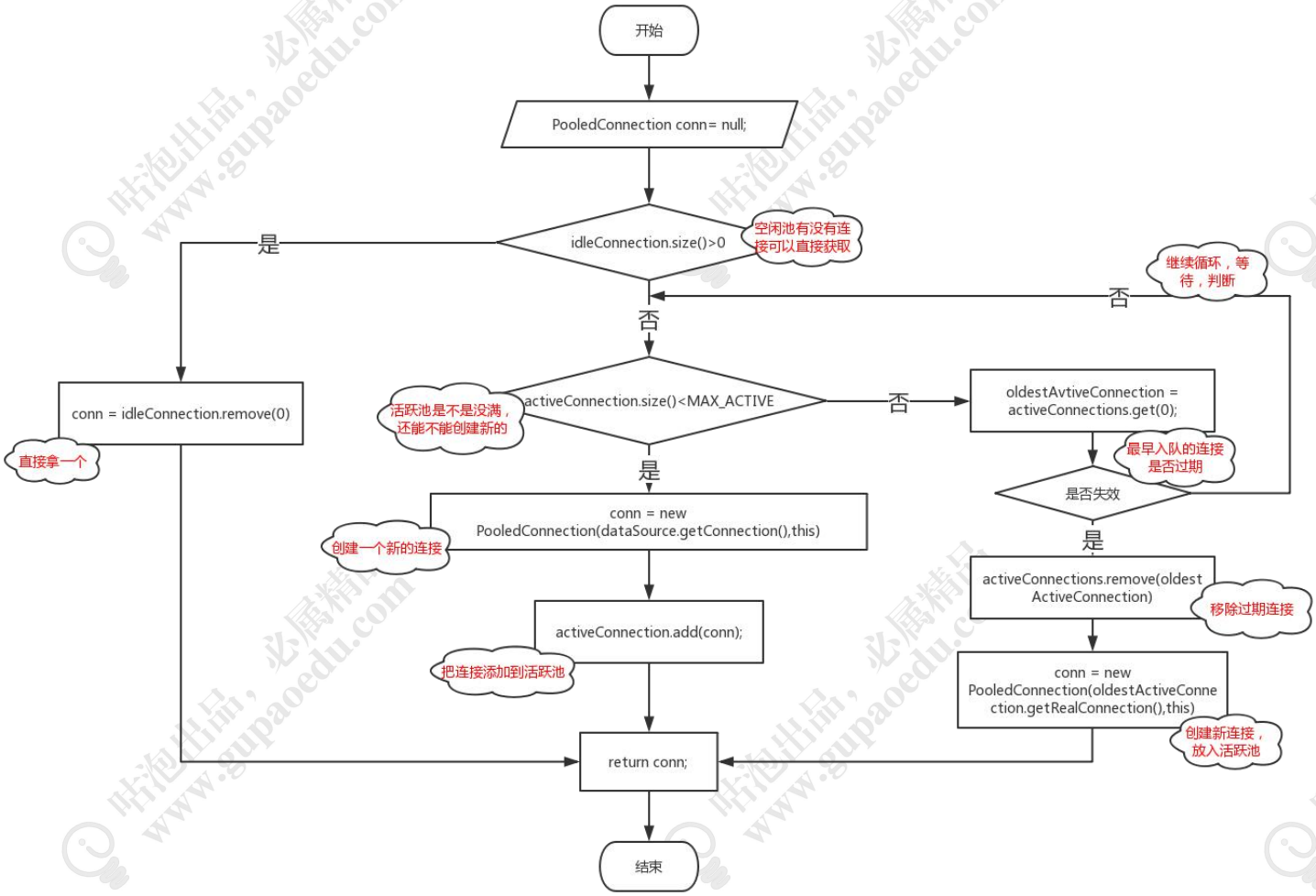
二级缓存：在同一个 namespace 共享，需要在 Mapper.xml 中开启，维护在 CachingExecutor 中。

8、MyBaits 支持哪些数据源类型？

UNPOOLED：不带连接池的数据源。

POOLED ： 带 连 接 池 的 数 据 源 ， 在 PooledDataSource 中 维 护 PooledConnection。

PooledDataSource 的 getConnection()方法流程图：



JNDI：使用容器的数据源，比如 Tomcat 配置了 C3P0。

自定义数据源：实现 DataSourceFactory 接口，返回一个 DataSource。

当 MyBatis 集成到 Spring 中的时候，使用 Spring 的数据源。

9、关联查询的延迟加载是怎么实现的？

动态代理（JAVASSIST、CGLIB），在创建实体类对象时进行代理，在调用代理对象的相关方法时触发二次查询。

10、MyBatis 翻页的几种方式和区别？

逻辑翻页：通过 RowBounds 对象。

物理翻页：通过改写 SQL 语句，可用插件拦截 Executor 实现。

11、怎么解决表字段变化引起的 MBG 文件变化的问题？

Mapper 继承：自动生成的部分不变，创建接口继承原接口，创建 MapperExt.xml。在继承接口和 MapperExt.xml 中修改。

通用 Mapper：提供支持泛型的通用 Mapper 接口，传入对象类型。

13、解析全局配置文件的时候，做了什么？

创建 Configuration，设置 Configuration

解析 Mapper.xml，设置 MappedStatement

14、没有实现类，MyBatis 的方法是怎么执行的？

MapperProxy 代理，代理类的 invoke()方法中调用了 SqlSession.selectOne()

15、接口方法和映射器的 statement id 是怎么绑定起来的？

（怎么根据接口方法拿到 SQL 语句的？）

MappedStatement 对象中存储了 statement 和 SQL 的映射关系

16、四大对象是什么时候创建的？

Executor：openSession()

StatementHandler、ResultsetHandler、ParameterHandler：

执行 SQL 时，在 SimpleExecutor 的 doQuery()中创建

17、ObjectFactory 的 create()方法什么时候被调用？

第一次被调用，创建 DefaultResultHandler 的时候：

DefaultResultSetHandler 类中：

handleResultSet　new DefaultResultHandler()

第二次被调用，处理结果集的时候：

DefaultResultSetHandler

handleResultSets——

handleRowValues——

handleRowValuesForSimpleResultMap——

getRowValue——

createResultObject——

createResultObject——

18、MyBatis 哪些地方用到了代理模式？

接口查找 SQL：MapperProxy

日志输出：ConnectionLogger、StatementLogger

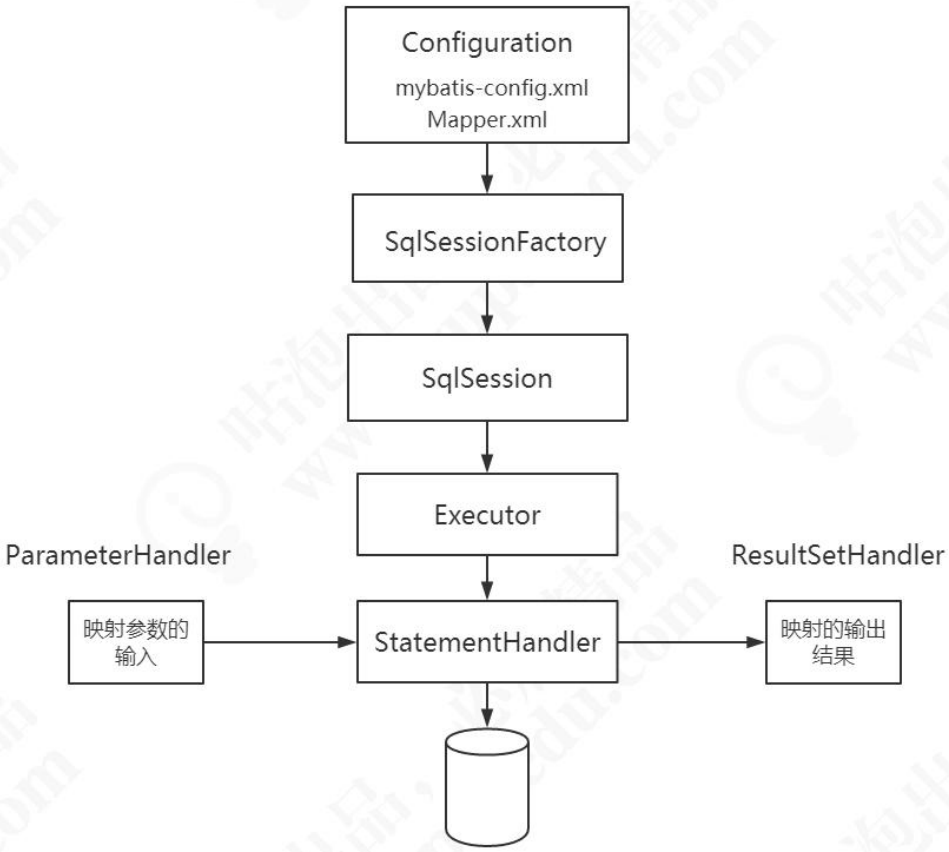
连接池：PooledDataSource 管理的 PooledConnection

延迟加载：ProxyFactory（JAVASSIST、CGLIB）

插件：Plugin

Spring 集成：SqlSessionTemplate 的内部类 SqlSessionInterceptor

19、MyBatis 主要的执行流程？涉及到哪些对象？



20、MyBatis 插件怎么编写和使用？原理是什么？（画图）

使用：继承 Interceptor 接口，加上注解，在 mybatis-config.xml 中配置

原理：动态代理，责任链模式，使用 Plugin 创建代理对象

在被拦截对象的方法调用的时候，先走到 Plugin 的 invoke()方法，再走到 Interceptor 实现类的 intercept()方法，最后通过 Invocation.proceed()方法调用被拦截对象的原方法

21、JDK 动态代理，代理能不能被代理？

能

22、MyBatis 集成到 Spring 的原理是什么？

SqlSessionTemplate 中有内部类 SqlSessionInterceptor 对 DefaultSqlSession 进行代理；

MapperFactoryBean 继承了 SqlSessionDaoSupport 获取 SqlSessionTemplate；

接口注册到 IOC 容器中的 beanClass 是 MapperFactoryBean。

23、DefaulSqlSession 和 SqlSessionTemplate 的区别是什么？

1）为什么 SqlSessionTemplate 是线程安全的？

其内部类 SqlSessionInterceptor 的 invoke()方法中的 getSqlSession()方法：

如果当前线程已经有存在的 SqlSession 对象，会在 ThreadLocal 的容器中拿到 SqlSessionHolder，获取 DefaultSqlSession。

如果没有，则会 new 一个 SqlSession，并且绑定到 SqlSessionHolder，放到 ThreadLocal 中。

SqlSessionTemplate 中在同一个事务中使用同一个 SqlSession。

调用 closeSqlSession()关闭会话时，如果存在事务，减少 holder 的引用计数。否则直接关闭 SqlSession。

2）在编程式的开发中，有什么方法保证 SqlSession 的线程安全？

SqlSessionManager 同时实现了 SqlSessionFactory、SqlSession 接口，通过 ThreadLocal 容器维护 SqlSession。

常见问题

用注解还是用 xml 配置？

常用注解：@Insert、@Select、@Update、@Delete、@Param、@Results、@Result

在 MyBatis 的工程中，我们有两种配置 SQL 的方式。一种是在 Mapper.xml 中集中管理，一种是在 Mapper 接口上，用注解方式配置 SQL。很多同学在工作中可能两种方式都用过。那到底什么时候用 XML 的方式，什么时候用注解的方式呢？

注解的缺点是 SQL 无法集中管理，复杂的 SQL 很难配置。所以建议在业务复杂的项

目中只使用 XML 配置的形式，业务简单的项目中可以使用注解和 XML 混用的形式。

Mapper 接口无法注入或 Invalid bound statement (not found)

我们在使用 MyBatis 的时候可能会遇到 Mapper 接口无法注入，或者 mapper statement id 跟 Mapper 接口方法无法绑定的情况。基于绑定的要求或者说规范，我们可以从这些地方去检查一下：

1、扫描配置，xml 文件和 Mapper 接口有没有被扫描到

2、namespace 的值是否和接口全类名一致

3、检查对应的 sql 语句 ID 是否存在

怎么获取插入的最新自动生成的 ID

在 MySQL 的插入数据使用自增 ID 这种场景 ,有的时候我们需要获得最新的自增 ID，比如获取最新的用户 ID。常见的做法是执行一次查询，max 或者 order by 倒序获取最大的 ID（低效、存在并发问题）。在 MyBatis 里面还有一种更简单的方式：

insert 成功之后，mybatis 会将插入的值自动绑定到插入的对象的 Id 属性中，我们用 getId 就能取到最新的 ID。

```
<insert id="insert" parameterType="com.gupaoedu.domain.Blog">
    insert into blog (bid, name, author_id)
    values (#{bid,jdbcType=INTEGER}, #{name,jdbcType=VARCHAR}, #{author,jdbcType=CHAR})
</insert>
```

```
blogService.addBlog(blog);
System.out.println(blog.getBid());
```

如何实现模糊查询 LIKE

1、字符串拼接

在 Java 代码中拼接%%（比如 name = "%" + name + "%"; ），直接 LIKE。因为没有预编译，存在 SQL 注入的风险，不推荐使用。

2、CONCAT（推荐）

```
<when test="empName != null and empName != ''">
    AND e.emp_name LIKE CONCAT(CONCAT('%', #{emp_name, jdbcType=VARCHAR}),'%')
</when>
```

3、bind 标签

```
<select id="getEmpList_bind" resultType="empResultMap" parameterType="Employee">
    <bind name="pattern1" value="'%' + empName + '%'" />
    <bind name="pattern2" value="'%' + email + '%'" />
    SELECT * FROM tbl_emp
    <where>
        <if test="empId != null">
            emp_id = #{empId,jdbcType=INTEGER},
        </if>
        <if test="empName != null and empName != ''">
            AND emp_name LIKE #{pattern1}
        </if>
        <if test="email != null and email != ''">
            AND email LIKE #{pattern2}
        </if>
    </where>
    ORDER BY emp_id
</select>
```

什么时候用#{}，什么时候用${}？

在 Mapper.xml 里面配置传入参数，有两种写法：#{} 、${}。作为 OGNL 表达式，

都可以实现参数的替换。这两种方式的区别在哪里？什么时候应该用哪一种？

要搞清楚这个问题，我们要先来说一下 PrepareStatement 和 Statement 的区别。

1、两个都是接口，PrepareStatement 是继承自 Statement 的；

2、Statement 处理静态 SQL，PreparedStatement 主要用于执行带参数的语句；

3、PreparedStatement 的 addBatch()方法一次性发送多个查询给数据库；

4、PS 相似 SQL 只编译一次（对语句进行了缓存，相当于一个函数）,比如语句相同参数不同，可以减少编译次数；

5、PS 可以防止 SQL 注入。

MyBatis 任意语句的默认值：PREPARED

这两个符号的解析方式是不一样的：

#会解析为 Prepared Statement 的参数标记符，参数部分用？代替。传入的参数会经过类型检查和安全检查。

（mybatis-standalone - MyBatisTest - testSelect()）

```
==>  Preparing: select * from blog where bid = ?
==> Parameters: 1(Integer)
<==    Columns: bid, name, author_id
<==        Row: 1, 咕泡学院, 1001
<==      Total: 1
查询结果：Blog{bid=1, name='咕泡学院', authorId='1001'}
```

$只会做字符串替换，比如参数是咕泡学院，结果如下：

（mybatis-standalone - MyBatisTest - selectBlogByBean ()）

```
==>  Preparing: select bid,name,author_id authorId from blog where name = '咕泡学院'
==> Parameters:
<==    Columns: bid, name, authorId
<==        Row: 1, 咕泡学院, 1001
<==      Total: 1
查询结果：[Blog{bid=1, name='咕泡学院', authorId='1001'}]
```

#和$的区别：

1、 是否能防止 SQL 注入：$方式不会对符号转义，不能防止 SQL 注入

2、 性能：$方式没有预编译，不会缓存

结论：

1、 能用#的地方都用#

2、 常量的替换，比如排序条件中的字段名称，不用加单引号，可以使用$

对象属性是基本类型 int double，数据库返回 null 是报错

使用包装类型。如 Integer，不要使用基本类型如 int。

If test !=null 失效了？

在实体类中使用包装类型。

XML 中怎么使用特殊符号，比如小于 &

1、转义< < （大于可以直接写）

2、使用<![CDATA[ ]]>——当 XML 遇到这种格式就会把[]里面的内容原样输出，不进行解析

作者：咕泡学院-青山

最后更新时间：2019 年 5 月 6 日 16:20:49