咕泡学院 JAVA VIP 高级课程教案

MYBATIS 源码分析系列专题

MyBatis 实用篇

主讲	James QQ 2904270631
版本	V2.0
适用对象	咕泡学院 JAVA 高级 VIP 学员

目录

咕泡学院 Java VIP 高级课程教案 0

MyBatis 源码分析系列专题 0

MyBatis 实用篇 0

第一节 MyBatis 介绍及使用 2

1. 认识 MyBatis （是什么都不知道就看源码就是掩耳盗铃） 2
2. 使用 MyBatis 2

第二节 配置文件解读 5

第一节 MyBatis 介绍及使用

1. 认识 MyBatis （是什么都不知道就看源码就是掩耳盗铃）

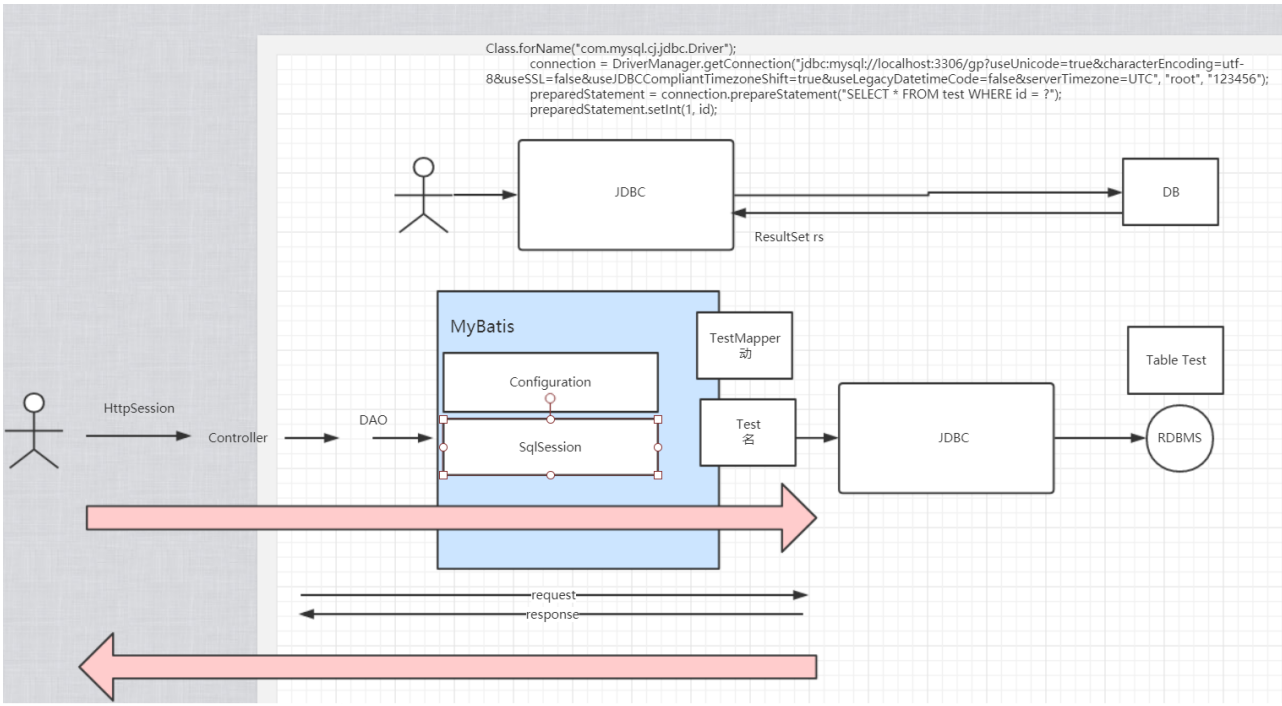
a) What is MyBatis? http://www.mybatis.org/mybatis-3/

MyBatis is a first class persistence framework with support for custom SQL, stored procedures and advanced mappings.是什么

MyBatis eliminates almost all of the JDBC code and manual setting of parameters and retrieval of results.优势

MyBatis can use simple XML or Annotations for configuration and map primitives, Map interfaces and Java POJOs (Plain Old Java Objects) to database records. 怎么做到的

b) 对比 JDBC 和 MyBatis



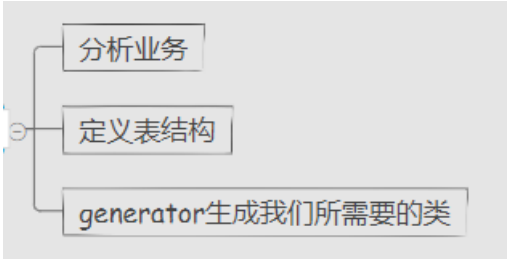
2. 使用 MyBatis

a) 使用过程

i. 编程式

ii. 集成式 managed 集成到 spring

iii. 工作当中的使用方式



iv. generator 使用步骤

1. pom.xml 配置 generator 插件

```
<plugin>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-maven-plugin</artifactId>
    <version>1.3.3</version>
    <configuration>
        <configurationFile>${project.basedir}/src/main/resources/mybatis/gen
    </configuration>
</plugin>
```

2. 配置 generatorConfig.xml
3. 执行 mvn mybatis-generator:generate
4. 生成 Bean 和 Example

v. 作用域 SCOPE 生命周期

	Scope
SqlSessionFactoryBuilder	method
SqlSessionFactory	application
SqlSession	request/method （可以认为是线程级）
Mapper	method

vi. Mapper 的 xml 和 annotation 形式

1. 兼容？ 兼容形式 → 互补
2. Pros vs cons

	Pros	Cons
Mapper.xml	● 跟接口分离、统一管理 ● 复杂的语句可以不影响接口的可读性	● 过多的 XML 文件
Annotation	● 接口就能看到 sql 语句，可读性高，不需要再去找 xml 文件，方便	● 复杂的联合查询不好维护，代码可读性差

vii. Config 文件部分解读 http://www.mybatis.org/mybatis-3/configuration.html

1. Environment
2. TypeHandler （java 和表字段类型的转换实现）
   a) 定义 com.gupao.dal.typehandlers.TestTypeHandle
   b) 注册 com.gupao.dal.config.MybatisConfig#localSessionFactoryBean
   c) 注册到使用字段上

i.
```
<resultMap id="BaseResultMap" type="com.gupao.dal.dao.Test">
  <id column="id" jdbcType="INTEGER" property="id" />
  <result column="nums" jdbcType="INTEGER" property="nums" />
  <result column="name" jdbcType="VARCHAR" property="name" typeHandler="com.gupao.dal.typehandlers.TestTypeHandle"/>
</resultMap>
```
取值

ii.
```
<insert id="insert" parameterType="com.gupao.dal.dao.Test" useGeneratedKeys="true" keyProperty="id">
  insert into test (id, nums, name
  )
  values (#{id,jdbcType=INTEGER}, #{nums,jdbcType=INTEGER}, #{name,jdbcType=VARCHAR,typeHandler=com.gupao.dal.typehandlers.TestTy...
  )
</insert>
```
赋值 插入

3. Plugins

   拦截范围

   - Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
   - ParameterHandler (getParameterObject, setParameters)
   - ResultSetHandler (handleResultSets, handleOutputParameters)
   - StatementHandler (prepare, parameterize, batch, update, query)

   a) 定义 com.gupao.dal.plugins.TestPlugin

   b) 注册 com.gupao.dal.config.MybatisConfig#localSessionFactoryBean

   c) 使用

第二节 配置文件解读

viii. mapper 文件解读

1. namespace

   关联到接口方法，区分类似 package 的作用

2. resultMap/resultType

	Pros	Cons
resultType	多表关联字段是清楚知道的，性能调优直观	创建很多实体类
resultMap	不需要写 join 语句	N+1 问题

3. sql
4. select insert update delete CRUD

   自增

```
<insert id="insert" parameterType="com.gupao.dal.dao.Test" useGeneratedKeys="true" keyProperty
```

5. 动态 SQL

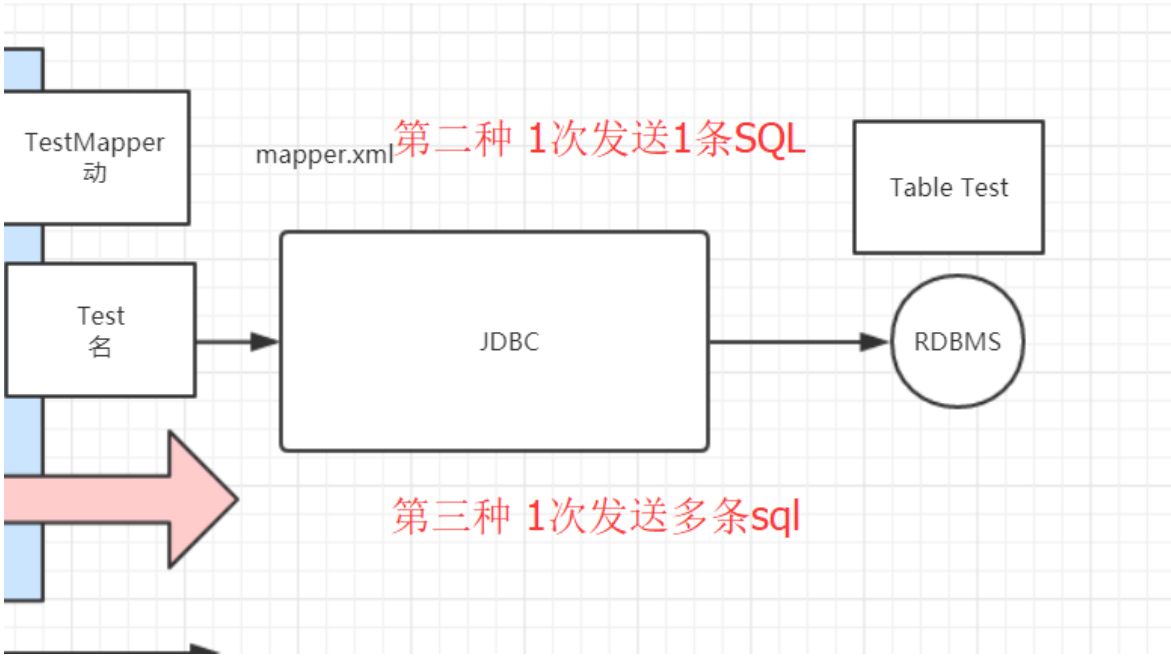
   http://www.mybatis.org/mybatis-3/dynamic-sql.html

6. 缓存
   a) 一级缓存
   b) 二级缓存

ix. Best practice

1. 分页
   a) 逻辑分页
      org.apache.ibatis.executor.resultset.DefaultResultSetHandler#handleRowValuesForSimpleResultMap 内存里分页
   b) 物理分页
      i. select … limit 0,10;
      ii. 分页插件
      https://github.com/pagehelper/Mybatis-PageHelper
2. 批量操作 Batch

	性能	
for 循环 一个一个插入	低 每次都要 IO	
foreach 拼 SQL（性能最高、推荐使用）	高	有 SQL 长度限制，定好 List 大小 show variables like '%packet%'; show variables like '%net_buffer%';
ExeutorType.BATCH		使用见代码 com.gupao.test.dal.TestMapperTest#insertBatchExType



3. 联合查询
   a) 嵌套结果
   b) 嵌套查询
      i. Lazy loading
      ii. com.gupao.dal.config.MybatisConfig#localSessionFactoryBean

```
SqlSessionFactory factory = sqlSessionFactoryBean.getObjec
//lazy loading switch
factory.getConfiguration().setLazyLoadingEnabled(true);
factory.getConfiguration().setAggressiveLazyLoading(false)
factory.getConfiguration().setProxyFactory(new CglibProxyF
```

APPENDIX

```
CREATE TABLE `posts` (
   `pid` int(11) NOT NULL AUTO_INCREMENT,
   `post_name` varchar(45) DEFAULT NULL,
   `blog_id` int(11) DEFAULT NULL,
   PRIMARY KEY (`pid`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `author` (
   `aid` int(11) NOT NULL AUTO_INCREMENT,
   `author_name` varchar(45) COLLATE utf8_bin DEFAULT NULL,
   PRIMARY KEY (`aid`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

```
CREATE TABLE `blog` (
   `bid` int(11) NOT NULL AUTO_INCREMENT,
   `name` varchar(45) COLLATE utf8_bin DEFAULT NULL,
   `author_id` int(11) DEFAULT NULL,
   PRIMARY KEY (`bid`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

```
CREATE TABLE `test` (
   `id` int(11) NOT NULL AUTO_INCREMENT,
   `nums` int(11) NOT NULL,
   `name` varchar(45) COLLATE utf8_bin DEFAULT NULL,
   PRIMARY KEY (`id`),
   KEY `idx_numbs` (`nums`)
) ENGINE=InnoDB AUTO_INCREMENT=4219 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```