😊1、mybatis简介

- 参考网站
  - http://www.mybatis.org/mybatis-3/zh/index.html
- 什么是 MyBatis
  - MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架
  - MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集
  - MyBatis 可以对配置和原生Map使用简单的 XML 或注解，将接口和 Java 的 POJOs 映射成数据库中的记录
- 安装
  - pom.xml 文件

```
<dependency>
 <groupId>org.mybatis</groupId>
 <artifactId>mybatis</artifactId>
 <version>x.x.x</version>
</dependency>
```

公众号java-mindmap

😊2、与hibernate对比

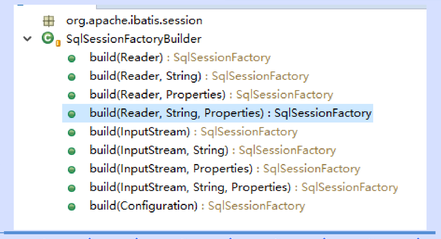
公众号java-mindmap

- 简介
  - mybatis
    - iBATIS 的着力点，则在于POJO 与SQL之间的映射关系
    - 然后通过映射配置文件，将SQL所需的参数，以及返回的结果字段映射到指定POJO
    - 相对Hibernate“O/R”而言，iBATIS 是一种“Sql Mapping”的ORM实现
  - Hibernate
    - 对数据库结构提供了较为完整的封装
    - Hibernate的O/R Mapping实现了POJO 和数据库表之间的映射，以及SQL 的自动生成和执行
    - Hibernate/OJB 会根据制定的存储逻辑，自动生成对应的SQL 并调用JDBC 接口加以执行
- 开发对比
  - mybatis
    - Mybatis框架相对简单很容易上手，但也相对简陋些
    - 针对高级查询，Mybatis需要手动编写SQL语句，以及ResultMap
  - Hibernate
    - Hibernate的真正掌握要比Mybatis来得难些
    - Hibernate有良好的映射机制，开发者无需关心SQL的生成与结果映射，可以更专注于业务流程。
- 调优方案
  - mybatis
    - 可以进行详细的SQL优化设计
    - 采用合理的Session管理机制；
  - Hibernate
    - 制定合理的缓存策略；
    - 尽量使用延迟加载特性；
    - 采用合理的Session管理机制；
    - 使用批量抓取，设定合理的批处理参数（batch_size）；
    - 进行合理的O/R映射设计
- 扩展性方面
  - mybatis
    - MyBatis项目中所有的SQL语句都是依赖所用的数据库的，所以不同数据库类型的支持不好
  - Hibernate
    - Hibernate与具体数据库的关联只需在XML文件中配置即可，所有的HQL语句与具体使用的数据库无关，移植性很好
- 缓存机制
  - mybatis
    - 默认情况下是没有开启缓存
    - 要开启二级缓存,你需要在你的 SQL 映射文件中添加一行: <cache/>
    - 1 映射语句文件中的所有 select 语句将会被缓存
    - 2 映射语句文件中的所有 insert,update 和 delete 语句会刷新缓存
    - 3 缓存会使用 Least Recently Used(LRU,最近最少使用的)算法来收回
    - 4 根据时间表(比如 no Flush Interval,没有刷新间隔), 缓存不会以任何时间顺序 来刷新。
    - 5 缓存会存储列表集合或对象(无论查询方法返回什么)的 1024 个引用。
    - 6 缓存会被视为是 read/write(可读/可写)的缓存,意味着对象检索不是共享的,而 且可以安全地被调用者修改,而不干扰其他调用者或线程所做的潜在修改。
  - Hibernate
    - 一级缓存是Session缓存，利用好一级缓存就需要对Session的生命周期进行管理好
    - 二级缓存是SessionFactory级的缓存，分为内置缓存和外置缓存
- 对比总结
  - mybatis
    - MyBatis可以进行更为细致的SQL优化，可以减少查询字段。
    - MyBatis容易掌握，而Hibernate门槛较高。
  - Hibernate
    - Hibernate的DAO层开发比MyBatis简单，Mybatis需要维护SQL和结果映射。
    - Hibernate对对象的维护和缓存要比MyBatis好，对增删改查的对象的维护要方便。
    - Hibernate数据库移植性很好，MyBatis的数据库移植性不好，不同的数据库需要写不同SQL。
    - Hibernate有更好的二级缓存机制，可以使用第三方缓存。MyBatis本身提供的缓存机制不佳。

😊3、几个关键类

公众号java-mindmap

- mybatis-config.xml
  - XML 配置文件（configuration XML）中包含了对 MyBatis 系统的核心设置，
  - 包含获取数据库连接实例的数据源（DataSource）和决定事务作用域和控制方式的事务管理器（TransactionManager）
- SqlSessionFactoryBuilder
  - 概念
    - SqlSessionFactoryBuilder通过类名就可以看出这个类的主要作用就是创建一个SqlSessionFactory，通过输入mybatis配置文件的字节流或者字符流，生成XMLConfigBuilder，XMLConfigBuilder创建一个Configuration，Configuration这个类中包含了mybatis的配置的一切信息，mybatis进行的所有操作都需要根据Configuration中的信息来进行。
  - 作用域（Scope）和生命周期
    - 可以重用 SqlSessionFactoryBuilder 来创建多个 SqlSessionFactory 实例，但是最好还是不要让其一直存在以保证所有的 XML 解析资源开放给更重要的事情
    - 这个类可以被实例化、使用和丢弃，一旦创建了 SqlSessionFactory，就不再需要它了。因此 SqlSessionFactoryBuilder 实例的最佳作用域是方法作用域（也就是局部方法变量）



- SqlSessionFactory接口
  - 概念
    - sql会话工厂，用于创建SqlSession
  - 作用域（Scope）和生命周期
    - SqlSessionFactory 一旦被创建就应该在应用的运行期间一直存在，没有任何理由对它进行清除或重建
    - 最佳作用域是应用作用域。有很多方法可以做到，最简单的就是使用单例模式或者静态单例模式。
  - 如何创建
    - 使用xml构建

```
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
```

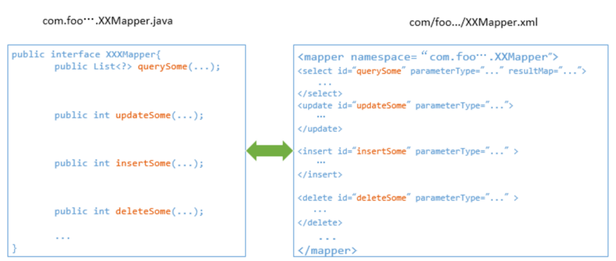
    - java代码构建

```
DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();
TransactionFactory transactionFactory = new JdbcTransactionFactory();
Environment environment = new Environment("development", transactionFactory, dataSource);
Configuration configuration = new Configuration(environment);
configuration.addMapper(BlogMapper.class);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(configuration);
```

- SqlSession接口
  - 概念
    - SqlSession是MyBatis的一个重要接口，定义了数据库的增删改查以及事务管理的常用方法。
    - SqlSession还提供了查找Mapper接口的有关方法。
  - 作用域（Scope）和生命周期
    - 每个线程都应该有它自己的 SqlSession 实例
    - SqlSession 的实例不是线程安全的，因此是不能被共享的，所以它的最佳的作用域是请求或方法作用域
    - 每次收到的 HTTP 请求，就可以打开一个 SqlSession，返回一个响应，就关闭它。
  - 如何创建

```
SqlSession session = sqlSessionFactory.openSession();
try {
 // do work
} finally {
 session.close();
}
```

- Mapper接口
  - 概念
    - 承载了实际的业务逻辑，其生命周期比较短，由SqlSession创建,用于将Java对象和实际的SQL语句对应起来。
    - Mapper接口是指程序员自行定义的一个数据操纵接口，类似于通常所说的DAO接口。跟DAO不同的地方在于Mapper接口只需要程序员定义，不需要程序员去实现，MyBatis会自动为Mapper接口创建动态代理对象。Mapper接口的方法通常与Mapper配置文件中的select、insert、update、delete等XML结点存在一一对应关系。
  - 实现方式
    - (1)使用XML配置文件的方式。
    - (2)使用注解方式。
    - (3直接使用MyBatis提供的API。
  - 如何创建



😊4、mybatis执行浅析

公众号java-mindmap

JDBC执行流程

mybatis在jdbc的基础上进行了封装

1. 加载JDBC驱动；
2. 建立并获取数据库连接；
3. 创建 JDBC Statements 对象；
4. 设置SQL语句的传入参数；
5. 执行SQL语句并获得查询结果；
6. 对查询结果进行转换处理并将处理结果返回；
7. 释放相关资源（关闭Connection，关闭Statement，关闭ResultSet）；

三层功能架构

http://www.jian...

API接口层

使用传统的MyBatis提供的API

List<?> | int | void　sqlSession.(select / selectList / selectMap / selectOne / update / delete / insert)(statementId [,parameterObject])

传统的MyBatis工作模式

Designed by LouLuan
http://blog.csdn.net/luanlouis

使用Mapper接口

- 接口中声明的方法和<mapper> 节点中的<select|update|delete|insert> 节点项对应
- 即<select|update|delete|insert> 节点的id值为Mapper 接口中的方法名称
- parameterType 值表示Mapper 对应方法的入参类型
- resultMap 值则对应了Mapper 接口表示的返回值类型或者返回结果集的元素类型

数据处理层

通过传入参数构建动态SQL语句

- MyBatis 通过传入的参数值，使用 Ognl 来动态地构造SQL语句，使得MyBatis 有很强的灵活性和扩展性
- 参数映射指的是对于java 数据类型和jdbc数据类型之间的转换

SQL语句的执行以及封装查询结果集成List<E>

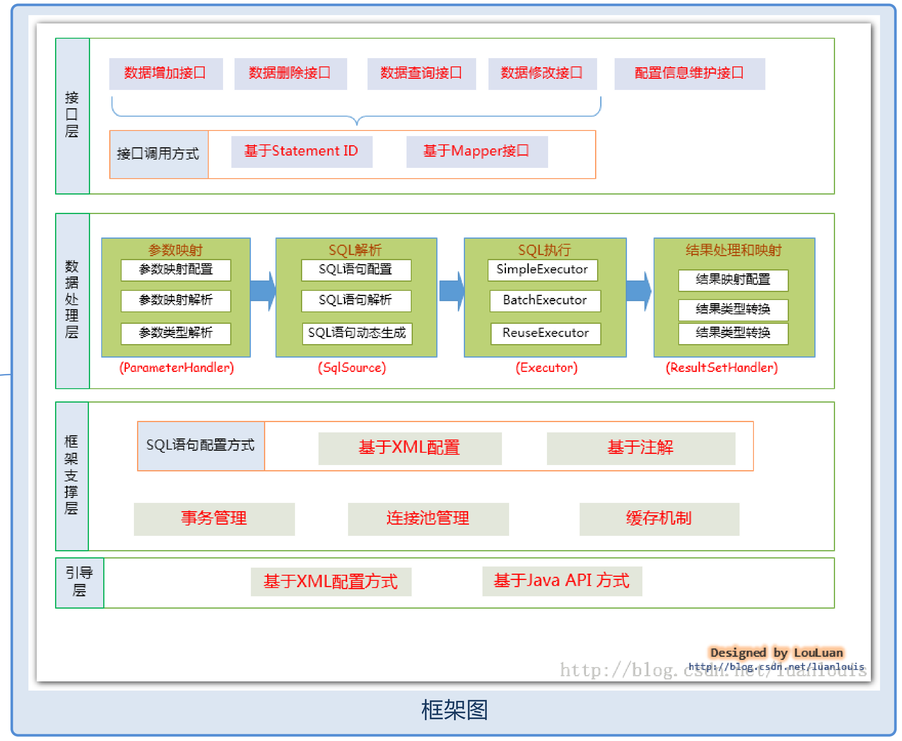
- 支持结果集关系一对多和多对一的转换
- 嵌套查询语句的查询
- 嵌套结果集的查询

基础支撑层

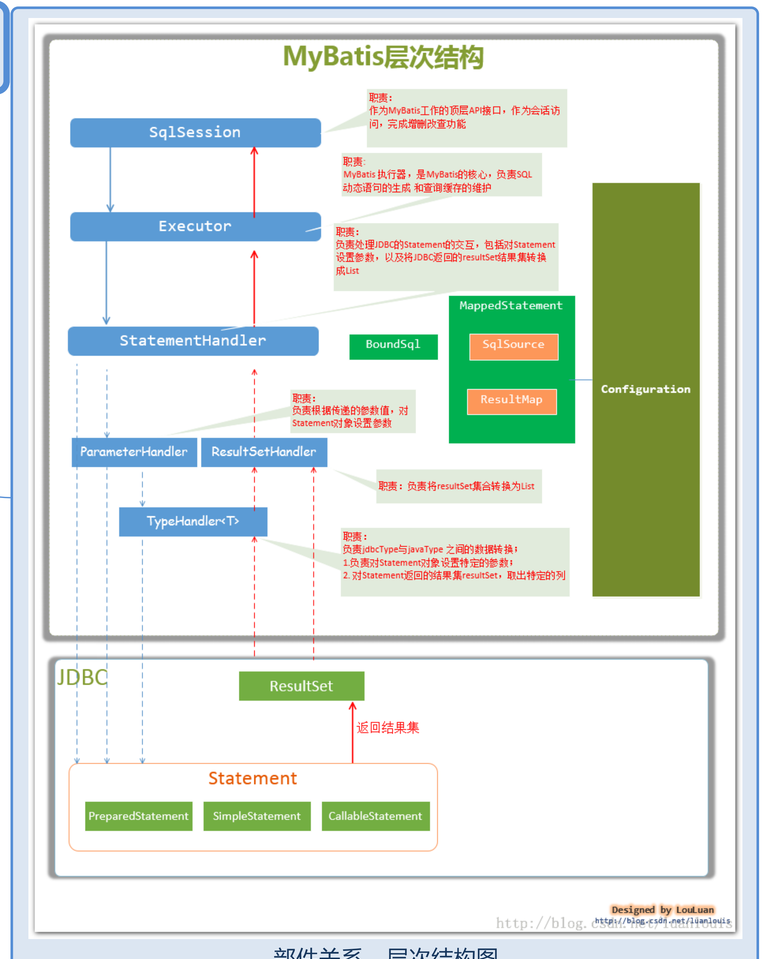
1. 事务管理机制
2. 连接池管理机制
3. 缓存机制
4. SQL语句的配置方式

😊5、MyBatis框架整体设计

公众号java-mindmap



框架图



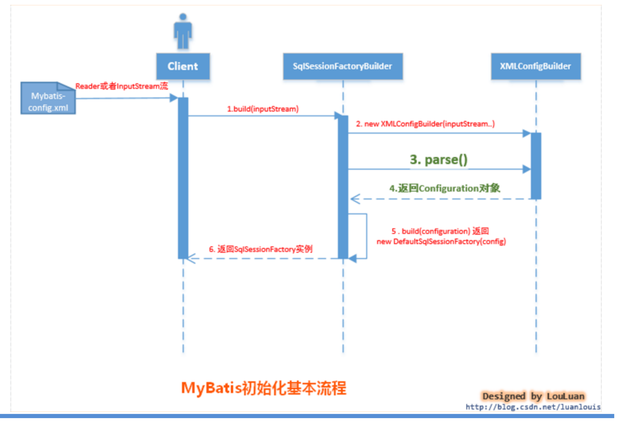
部件关系，层次结构图

😊6、mybatis初始化与执行sql过程

公众号java-mindmap

初始化

- 概念：可以这么说，MyBatis初始化的过程，就是创建 Configuration对象的过程
- 过程
  - 1 加载配置文件mybatis-config.xml到MyBatis内部
    - configuration 配置
      - properties 属性
      - settings 设置
      - typeAliases 类型命名
      - typeHandlers 类型处理器
      - objectFactory 对象工厂
      - plugins 插件
      - environments 环境
        - environment 环境变量
          - transactionManager 事务管理器
          - dataSource 数据源
      - databaseIdProvider 数据库厂商标识
      - mappers 映射器
  - 2 使用 org.apache.ibatis.session.Configuration对象作为一个所有配置信息的容器，Configuration对象的组织结构和XML配置文件的组织结构几乎完全一样，这样配置文件的信息就存到了Configuration这个类中了
- 流程图



sql执行总体流程

- 加载配置并初始化：加载配置文件，将SQL的配置信息加载成为一个个MappedStatement对象（包括了传入参数映射配置、执行的SQL语句、结果映射配置），存储在内存中
- 接收调用请求
  - 触发条件：调用Mybatis提供的API
  - 传入参数：为SQL的ID和传入参数对象
  - 处理过程：将请求传递给下层的请求处理层进行处理。
- 处理操作请求
  - 触发条件：API接口层传递请求过来
  - 传入参数：为SQL的ID和传入参数对象
  - 处理过程：
    - (A)根据SQL的ID查找对应的MappedStatement对象。
    - (B)根据传入参数对象解析MappedStatement对象，得到最终要执行的SQL和执行传入参数。
    - (C)获取数据库连接，根据得到的最终SQL语句和执行传入参数到数据库执行，并得到执行结果。
    - (D)根据MappedStatement对象中的结果映射配置对得到的执行结果进行转换处理，并得到最终的处理结果
    - (E)释放连接资源。
    - (4)返回处理结果将最终的处理结果返回。
- 整体执行流程图.jpg

小结过程

加载配置 → SQL解析 → SQL执行 → 结果映射

😊7、mybatis源码的几个主要部件

公众号java-mindmap

- SqlSession：作为MyBatis工作的主要顶层API，表示和数据库交互的会话，完成必要数据库增删改查功能；
- Executor：MyBatis执行器，是MyBatis 调度的核心，负责SQL语句的生成和查询缓存的维护；
- StatementHandler：封装了JDBC Statement操作，负责对JDBC statement 的操作，如设置参数、将Statement结果集转换成List集合。
- ParameterHandler：负责对用户传递的参数转换成JDBC Statement 所需要的参数；
- ResultSetHandler：负责将JDBC返回的ResultSet结果集对象转换成List类型的集合；
- TypeHandler：负责java数据类型和jdbc数据类型之间的映射和转换；
- MappedStatement：MappedStatement维护了一条<select|update|delete|insert>节点的封装；
- SqlSource：负责根据用户传递的parameterObject，动态地生成SQL语句，将信息封装到BoundSql对象中，并返回；
- BoundSql：表示动态生成的SQL语句以及相应的参数信息；
- Configuration：MyBatis所有的配置信息都维持在Configuration对象之中；

😊8、XML 映射配置文件（一）

公众号java-mindmap

properties

http://www.myb...

- 作用：引用propertis文件并读取配置信息，也可以在<properties/>标签中定义属性
- 例子
  - 1 创建一个资源文件jdbc.properties

```
jdbc.driverClassName=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@localhost:1521:orcl
jdbc.username=mybatis
jdbc.password=mybatis
```

  - 2 mybatis-config.xml中引入：<properties resource="jdbc.properties" />
  - 3 也可以在<properties/>标签中定义属性

```
<properties resource="jdbc.properties">
   <property name="jdbc.driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
</properties>
```

  - 4 使用properties文件里的属性

```
<dataSource type="POOLED">
 <property name="driver" value="${jdbc.driverClassName}" />
  <property name="url" value="${jdbc.url}" />
  <property name="username" value="${jdbc.username}" />
  <property name="password" value="${jdbc.password}" />
</dataSource>
```

- 配置的加载顺序
  - 1 在 properties 元素体内指定的属性首先被读取。
  - 2 然后根据 properties 元素中的 resource 属性读取类路径下属性文件或根据 url 属性指定的路径读取属性文件，并覆盖已读取的同名属性。
  - 3 最后读取作为方法参数传递的属性，并覆盖已读取的同名属性。

settings

- 概念：MyBatis 中极为重要的调整设置，它们会改变 MyBatis 的运行时行为
- 配置方式

```
<settings>
<setting name="cacheEnabled" value="true"/>
<setting name="lazyLoadingEnabled" value="true"/>
<setting name="multipleResultSetsEnabled" value="true"/>
<setting name="useColumnLabel" value="true"/>
<setting name="useGeneratedKeys" value="false"/>
<setting name="autoMappingBehavior" value="PARTIAL"/>
<setting name="autoMappingUnknownColumnBehavior" value="WARNING"/>
<setting name="defaultExecutorType" value="SIMPLE"/>
<setting name="defaultStatementTimeout" value="25"/>
<setting name="defaultFetchSize" value="100"/>
<setting name="safeRowBoundsEnabled" value="false"/>
<setting name="mapUnderscoreToCamelCase" value="false"/>
<setting name="localCacheScope" value="SESSION"/>
<setting name="jdbcTypeForNull" value="OTHER"/>
<setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>
</settings>
```

- 常用
  - cacheEnabled：该配置影响的所有映射器中配置的缓存的全局开关。默认true
  - lazyLoadingEnabled：延迟加载的全局开关。当开启时，所有关联对象都会延迟加载。 特定关联关系中可通过设置fetchType属性来覆盖该项的开关状态。默认false
  - aggressiveLazyLoading：当开启时，任何方法的调用都会加载该对象的所有属性。否则，每个属性会按需加载。默认false (true in ≤3.4.1)
  - logImpl：指定 MyBatis 所用日志的具体实现，未指定时将自动查找。

😊9、XML 映射配置文件（二）

公众号java-mindmap

typeAliases

- 概念
  - 类型别名。
  - 给java类型取一个别名，方便在核心配置、映射配置中来使用这个java类型。
  - 它只和 XML 配置有关，存在的意义仅在于用来减少类完全限定名的冗余
- 比较（区别看resultType）
  - 不使用别名

```
<select id="findUserById" resultType="com.yu.model.User" parameterType="long">
  select * from t_user where id = #{id}
</select>
```

  - 使用别名
    - 方式一
      - 在核心配置文件中加上配置

```
<typeAliases>
 <typeAlias type="com.yu.model.User" alias="User" />
 <typeAlias type="com.yu.model.Org" alias="Org"/>
</typeAliases>
```

    - 方式二

```
<typeAliases>
 <package name="com.yu.model"/>
</typeAliases>
```

      - 或者注解方式

```
@Alias("user")
public class User {
  ...
}
```

        - 在没有注解的情况下，会使用 Bean 的首字母小写的非限定类名来作为它的别名
        - 若有注解，则别名为其注解值
    - 修改UserMapper.xml的配置

```
<select id="findUserById" resultType="User" parameterType="long">
  select * from t_user where id = #{id}
</select>
```

- 内置类型别名
  - 已经为许多常见的 Java 类型内建了相应的类型别名。它们都是大小写不敏感的，需要注意的是由基本类型名称重复导致的特殊处理。
  - 内置类型别名.png

typeHandlers

- 概念
  - 类型处理器
  - 无论是 MyBatis 在预处理语句（PreparedStatement）中设置一个参数时，还是从结果集中取出一个值时， 都会用类型处理器将获取的值以合适的方式转换成 Java 类型
- 内置类处理器
  - 内置类处理器.png
- 用途
  - 1）获取数据库的值，以合适的方式转变为对应的java类型
  - 2）将java类型，以合适的方式转化为数据库的保存类型
- 自定义类型处理器
  - 实现 org.apache.ibatis.type.TypeHandler 接口
  - 或继承一个很便利的类 org.apache.ibatis.type.BaseTypeHandler
  - 然后可以选择性地将它映射到一个 JDBC 类型

```
// ExampleTypeHandler.java
@MappedJdbcTypes(JdbcType.VARCHAR)
public class ExampleTypeHandler extends BaseTypeHandler<String> {

 @Override
public void setNonNullParameter(PreparedStatement ps, int i, String
parameter, JdbcType jdbcType) throws SQLException {
 ps.setString(i, parameter);
 }

 @Override
public String getNullableResult(ResultSet rs, String columnName) throws
SQLException {
 return rs.getString(columnName);
 }

 @Override
public String getNullableResult(ResultSet rs, int columnIndex) throws
SQLException {
 return rs.getString(columnIndex);
 }

 @Override
public String getNullableResult(CallableStatement cs, int columnIndex) throws
SQLException {
 return cs.getString(columnIndex);
 }
}
```

  - 例子

```
<!-- mybatis-config.xml -->
<typeHandlers>
<typeHandler handler="org.mybatis.example.ExampleTypeHandler"/>
</typeHandlers>
```

    - 使用这个的类型处理器将会覆盖已经存在的处理 Java 的 String 类型属性和 VARCHAR 参数及结果的类型处理器。
- 处理枚举类型
  - 若想映射枚举类型 Enum，则需要从 EnumTypeHandler 或者 EnumOrdinalTypeHandler 中选一个来使用
  - 默认情况下，MyBatis 会利用 EnumTypeHandler 来把 Enum 值转换成对应的名字。

```
<!-- mybatis-config.xml -->
<typeHandlers>
<typeHandler handler="org.apache.ibatis.type.EnumOrdinalTypeHandler" javaType="java.math.RoundingMode"/>
</typeHandlers>
```

😊10、XML 映射配置文件（三）

公众号java-mindmap

插件（plugins）

MyBatis 允许你在已映射语句执行过程中的某一点进行拦截调用

允许使用插件来拦截的方法调用包括

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

方法

只需实现 Interceptor 接口，并指定了想要拦截的方法签名即可

例子

拦截在 Executor 实例中所有的 “update” 方法调用

```
// ExamplePlugin.java
@Intercepts({@Signature(
 type= Executor.class,
 method = "update",
args = {MappedStatement.class,Object.class})})
public class ExamplePlugin implements Interceptor {
public Object intercept(Invocation invocation) throws Throwable {
 return invocation.proceed();
 }
public Object plugin(Object target) {
 return Plugin.wrap(target, this);
 }
public void setProperties(Properties properties) {
 }
 }
```

```
<!-- mybatis-config.xml -->
<plugins>
<plugin interceptor="org.mybatis.example.ExamplePlugin">
 <property name="someProperty" value="100"/>
 </plugin>
</plugins>
```

databaseIdProvider

可以根据不同的数据库厂商执行不同的语句，这种多厂商的支持是基于映射语句中的 databaseId 属性

MyBatis 会加载不带 databaseId 属性和带有匹配当前数据库 databaseId 属性的所有语句

```
<databaseIdProvider type="DB_VENDOR" />
```

```
<databaseIdProvider type="DB_VENDOR">
 <property name="SQL Server" value="sqlserver"/>
    <property name="DB2" value="db2"/>
 <property name="Oracle" value="oracle" />
</databaseIdProvider>
```

映射器（mappers）

概念：用于引用定义好的映射定义，告诉mybatis去哪里找我们的sql定义配置

四中方式

1. 直接引用xml文件

```
<mappers>
 <mapper resource="com/yu/res/UserMapper.xml" />
</mappers>
```

2. 通过绝对路径引用，注意在绝对路径前加上：“file:///”

```
<mappers>
 <mapper url="file:///D:/workspace/mywork/MyBatis/src/com/yu/res/UserMapper.xml"/>
</mappers>
```

3. 引用mapper接口对象的方式

```
<mappers>
  <mapper resource="com.yu.mapper.UserMapper" />
</mappers>
```

4. 引用mapper接口包的方式

```
<mappers>
 <package name="com.yu.mapper"/>
</mappers>
```

配置环境（environments）

配置成适应多种环境，这种机制有助于将 SQL 映射应用于多种数据库之中

不过要记住：尽管可以配置多个环境，每个 SqlSessionFactory 实例只能选择其一。

使用场景

1）为了开发设置不同的数据库配置

2）测试和生产环境数据库不同

3）有多个数据库却共享相同的模式，即对不同的数据库使用相同的SQL映射

```
<!-- 对事务的管理和连接池的配置 -->
<environments default="oracle_jdbc">
  <environment id="oracle_jdbc">
    <transactionManager type="JDBC" />
    <dataSource type="POOLED">
    <property name="driver" value="${jdbc.oracle.driverClassName}" />
     <property name="url" value="${jdbc.oracle.url}" />
    <property name="username" value="${jdbc.oracle.username}" />
     <property name="password" value="${jdbc.oracle.password}" />
    </dataSource>
  </environment>

  <environment id="mysql_jdbc">
    <transactionManager type="JDBC" />
    <dataSource type="POOLED">
    <property name="driver" value="${jdbc.mysql.driverClassName}" />
     <property name="url" value="${jdbc.mysql.url}" />
    <property name="username" value="${jdbc.mysql.username}" />
     <property name="password" value="${jdbc.mysql.password}" />
    </dataSource>
  </environment>
</environments>
```

代码：用default指定默认的数据库链接：：<environments default="oracle_jdbc">

根据数据库环境，获取SqlSessionFactory

```
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, environment);
```

```
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, environment,properties);
```

environment中的配置

transactionManager

事务管理器

例子：<transactionManager type="JDBC" />

type取值

- JDBC：简单的使用JDBC的提交和回滚设置，一览与从数据员得到的链接来管理事务范围
- MANAGED：这个配置几乎什么都没做，它从来不提交或回滚一个链接，而它让容器来管理事务的整个生命周期

dataSource

数据源

例子

```
<dataSource type="POOLED">
<property name="driver" value="${jdbc.mysql.driverClassName}" />
 <property name="url" value="${jdbc.mysql.url}" />
<property name="username" value="${jdbc.mysql.username}" />
 <property name="password" value="${jdbc.mysql.password}" />
</dataSource>
```

type取值

- UNPOOLED：这个数据源的实现是每次被请求时打开和关闭连接。速度会有一些慢，适用于简单的应用程序。
- POOLED：这是JDBC链接对象的数据源连接池的实现，用来避免创建新的链接实例时必要的连接和认证时间。适用于当前Web应用程序用来快速响应请求
- JNDI：这个数据源是为了使用如Spring或应用服务器这类的容器，容器可以集中或在外部配置数据源，然后设置JNDI上下文的引用