# Mybatis和hibernate区别和应用场景

Mybatis:使开发者只专注于SQL的半自动化框架,容易入门，学习成本低。

Hibernate:是一个标准的ORM（对象关系模型）框架，入门门槛较高。是一个全自动化的框架，开发者只需要操纵对象无需编写SQL。缺点：SQL无法控制对于复杂的查询由于无法优化SQL查询效率较低。

应用场景：适用于项目需求变化较少的中小型项目，如：ERP、OA项目

# Dao层传统开发方法

需程序员开发DAO接口和实现类，并将SqlSessionFactory注入到实现类中。

# Dao层Mapper代理开发方法

只需要编写Mapper.xml和Mapper.java接口，实现类由Mybatis自动生成代理类。但是需遵循一些规范。

|  |
| --- |
| **Mapper代理开发规范：**   1. Mapper.xml中namespace就是Mapper.java类的全路径。 2. Mapper.xml中的statement id和Mapper.java类中的方法名称一致。 3. Mapper.xml中的parameterType和Mapper.java类中的输入参数一致。 4. Mapper.xml中的resultType和Mapper.java类中的输出参数一致。 |

# SQLMapConfig.xml配置

## Properties

在SQLMapConfig.xml将配置文件引入：

|  |
| --- |
| <properties resource="org/mybatis/example/db.properties">  </properties> |

db.properties配置文件

|  |
| --- |
| jdbc.driver=oracle.jdbc.driver.OracleDriver  jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:SUNCPS  jdbc.username=SUNCPS  jdbc.password=111111 |

在SQLMapConfig.xml引入配置文件中的变量

|  |
| --- |
| <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>  </environments> |

## Settiongs

## Typealiases(自定义别名)

类型别名是为Java 类型命名一个短的名字。它只和XML 配置有关，只用来减少类完全

限定名的多余部分。

单个定义别名：

|  |
| --- |
| <typeAliases>  <typeAlias alias="Tag" type="domain.blog.Tag"/>  </typeAliases> |

批量定义别名：

|  |
| --- |
| <typeAliases>  <!—mybatis 自动扫描包下的po类 自动定义别名为类名首字母大小写都可以-->  <package name=”domain.blog”>  </typeAliases> |

## typeHandlers(类型处理器)

无论是MyBatis 在预处理语句中设置一个参数，还是从结果集中取出一个值时，类型处理器被用来将**获取的值以合适的方式转换成Java 类型**。下面这个表格描述了默认的类型处理器。

类型处理器 Java 类型 JDBC 类型

BooleanTypeHandler Boolean ，boolean 任何兼容的布尔值

ByteTypeHandler Byte，byte 任何兼容的数字或字节类型

ShortTypeHandler Short，short 任何兼容的数字或短整型

IntegerTypeHandler Integer ，int 任何兼容的数字和整型

LongTypeHandler Long ，long 任何兼容的数字或长整型

FloatTypeHandler Float ，float 任何兼容的数字或单精度浮点型

DoubleTypeHandler Double，double 任何兼容的数字或双精度浮点型

BigDecimalTypeHandler BigDecimal 任何兼容的数字或十进制小数类型

StringTypeHandler String CHAR 和VARCHAR 类型

ClobTypeHandler String CLOB 和LONGVARCHAR 类型

NStringTypeHandler String NVARCHAR 和NCHAR 类型

NClobTypeHandler String NCLOB 类型

ByteArrayTypeHandler byte[] 任何兼容的字节流类型

BlobTypeHandler byte[] BLOB 和LONGVARBINARY 类型

DateTypeHandler Date （java.util ） TIMESTAMP 类型

DateOnlyTypeHandler Date （java.util ） DATE 类型

TimeOnlyTypeHandler Date （java.util ） TIME 类型

SqlTimestampTypeHandler Timestamp （java.sql ） TIMESTAMP 类型

SqlDateTypeHandler Date （java.sql ） DATE 类型

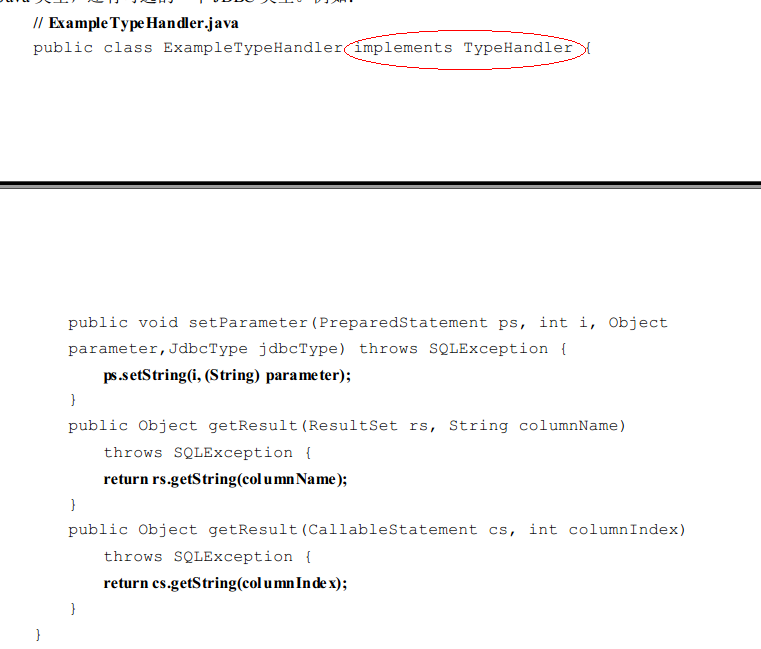
SqlTimeTypeHandler Time （java.sql ） TIME 类型

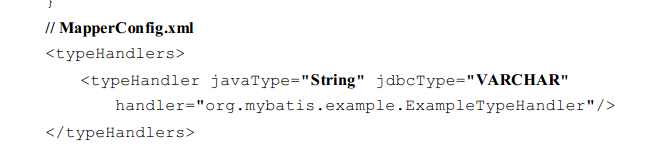
ObjectTypeHandler Any 其他或未指定类型

EnumTypeHandler Enumeration 类型 VARCHAR-任何兼容的字符串类型，

作为代码存储（而不是索引）。

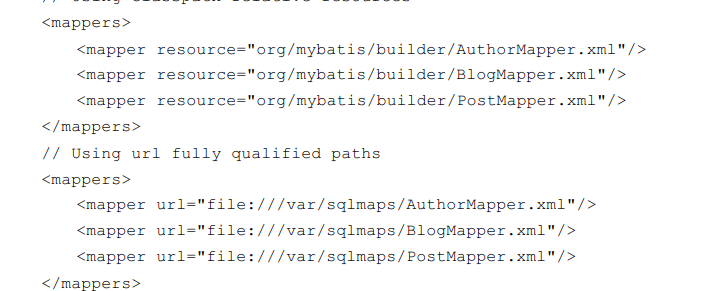
自定义处理器：





## mappers

单个加载



批量加载

|  |
| --- |
| <mappers>  <package name=”domain.blog”>  </mappers> |

# Mapper.xml配置

## Parametertype

### 传递简单类型

### 传递POJO对象

### 传递POJO包装对象

需求：有时可能传入的查询条件很复杂，涉及到的查询条件包含多个类的信息，这时建议使用包装对象。

#### 定义包装对象

#### Mapper.xml配置文件

## Resultype

## ResultMap

## #{}、${}

## 获得主键

## 查询

## 增加、更新、删除

## 动态SQL

### Where If

|  |
| --- |
| <select id=”findActiveBlogLike” parameterType=”Blog” resultType=”Blog”>  SELECT \* FROM BLOG  <where>  <if test=”state != null”>  state = #{state}  </if>  <if test=”title != null”>  AND title like #{title}  </if>  <if test=”author != null and author.name != null”>  AND title like #{author.name}  </if>  </where>  </select> |

### Foreach

|  |
| --- |
| <select id="selectPostIn" resultType="domain.blog.Post">  SELECT \* FROM POST P  WHERE ID in  <foreach item="item" index="index" collection="list"  open="(" separator="," close=")">  #{item}  </foreach>  </select> |

Collection:指定集合

Item：迭代到指定的变量

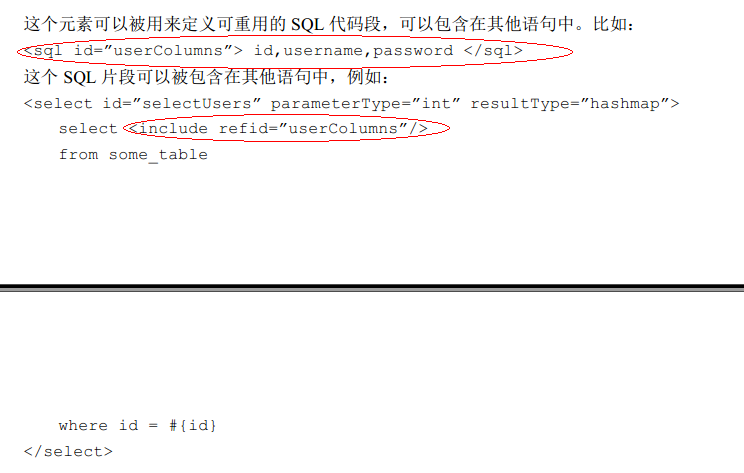
Index:序号

Open：开始字符串

Close：结束字符串

Separator：分隔符

## SQL片段



# mybatis逆向工程

## 下载逆向工程jar包和配置文件



## 加入数据库驱动包



## 修改配置文件generatorConfig.xml



进入到该目录运行如下命令：

java -jar mybatis-generator-core-1.3.2.jar -configfile generatorConfig.xml -overwrite