# 逆向工程

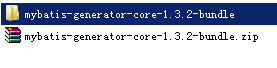
## 什么是逆向工程

mybaits需要程序员自己编写sql语句，mybatis官方提供逆向工程 可以针对单表自动生成mybatis执行所需要的代码（mapper.java,mapper.xml、po..）

企业实际开发中，常用的逆向工程方式：

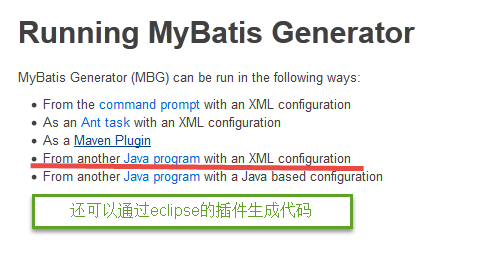
由于数据库的表生成java代码。

## 下载逆向工程



## 使用方法（会用）

### 运行逆向工程



建议使用java程序方式，不依赖开发工具。

### 生成代码配置文件

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

<!DOCTYPE generatorConfiguration

PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

<generatorConfiguration>

<context id=*"testTables"* targetRuntime=*"MyBatis3"*>

<commentGenerator>

<!-- 是否去除自动生成的注释 true：是 ： false:否 -->

<property name=*"suppressAllComments"* value=*"true"* />

</commentGenerator>

<!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->

<jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*

connectionURL=*"jdbc:mysql://localhost:3306/mybatis"* userId=*"root"*

password=*"mysql"*>

</jdbcConnection>

<!-- <jdbcConnection driverClass="oracle.jdbc.OracleDriver"

connectionURL="jdbc:oracle:thin:@127.0.0.1:1521:yycg"

userId="yycg"

password="yycg">

</jdbcConnection> -->

<!-- 默认false，把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer，为 true时把JDBC DECIMAL 和

NUMERIC 类型解析为java.math.BigDecimal -->

<javaTypeResolver>

<property name=*"forceBigDecimals"* value=*"false"* />

</javaTypeResolver>

<!-- targetProject:生成PO类的位置 -->

<javaModelGenerator targetPackage=*"cn.itcast.ssm.po"*

targetProject=*".\src"*>

<!-- enableSubPackages:是否让schema作为包的后缀 -->

<property name=*"enableSubPackages"* value=*"false"* />

<!-- 从数据库返回的值被清理前后的空格 -->

<property name=*"trimStrings"* value=*"true"* />

</javaModelGenerator>

<!-- targetProject:mapper映射文件生成的位置 -->

<sqlMapGenerator targetPackage=*"cn.itcast.ssm.mapper"*

targetProject=*".\src"*>

<!-- enableSubPackages:是否让schema作为包的后缀 -->

<property name=*"enableSubPackages"* value=*"false"* />

</sqlMapGenerator>

<!-- targetPackage：mapper接口生成的位置 -->

<javaClientGenerator type=*"XMLMAPPER"*

targetPackage=*"cn.itcast.ssm.mapper"*

targetProject=*".\src"*>

<!-- enableSubPackages:是否让schema作为包的后缀 -->

<property name=*"enableSubPackages"* value=*"false"* />

</javaClientGenerator>

<!-- 指定数据库表 -->

<table tableName=*"items"*></table>

<table tableName=*"orders"*></table>

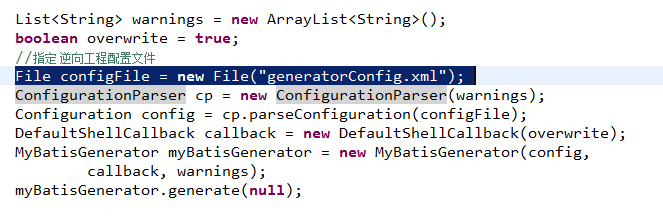
<table tableName=*"orderdetail"*></table>

<table tableName=*"user"*></table>

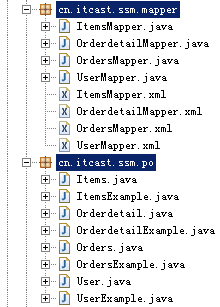
</context>

</generatorConfiguration>

### 执行生成程序



生成后的代码：



### 使用生成的代码

需要将生成工程中所生成的代码拷贝到自己的工程中。

测试ItemsMapper中的方法

//自定义条件查询

@Test

**public** **void** testSelectByExample() {

ItemsExample itemsExample = **new** ItemsExample();

//通过criteria构造查询条件

ItemsExample.Criteria criteria = itemsExample.createCriteria();

criteria.andNameEqualTo("笔记本3");

//可能返回多条记录

List<Items> list = itemsMapper.selectByExample(itemsExample);

System.*out*.println(list);

}

//根据主键查询

@Test

**public** **void** testSelectByPrimaryKey() {

Items items = itemsMapper.selectByPrimaryKey(1);

System.*out*.println(items);

}

//插入

@Test

**public** **void** testInsert() {

//构造 items对象

Items items = **new** Items();

items.setName("手机");

items.setPrice(999f);

itemsMapper.insert(items);

}

//更新数据

@Test

**public** **void** testUpdateByPrimaryKey() {

//对所有字段进行更新，需要先查询出来再更新

Items items = itemsMapper.selectByPrimaryKey(1);

items.setName("水杯");

itemsMapper.updateByPrimaryKey(items);

//如果传入字段不空为才更新，在批量更新中使用此方法，不需要先查询再更新

//itemsMapper.updateByPrimaryKeySelective(record);

}