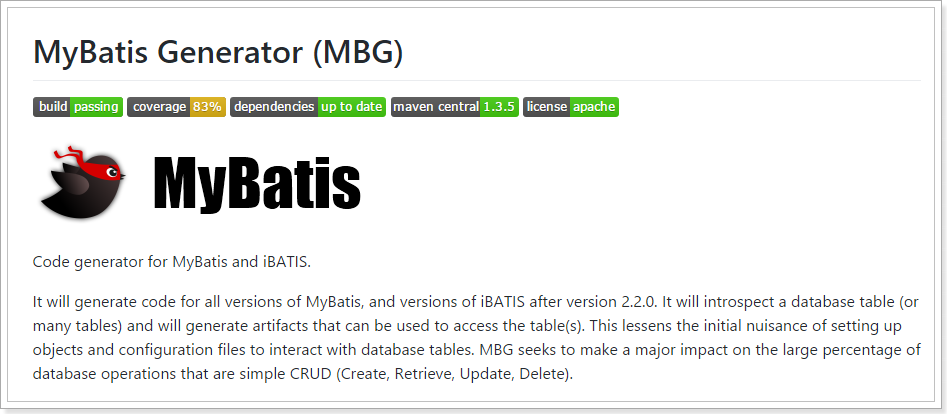
# MyBatis Generator (逆向工程)

## MyBatis Generator 概述

MyBatis的官方提供了MyBatis的逆向工程高级：MyBatis Generator（以下简称为MBG），可以根据数据库表生成：实体类、对应Mapper接口、对应的Mapper.xml映射文件。

MBG的解决了对数据库操作有最频繁的一些简单的单表CRUD（插入，查询，更新，删除）操作。 您仍然需要对联合查询和存储过程手写SQL和对象。



英文文档首页：<http://www.mybatis.org/generator/>

中文翻译文档：<http://mbg.cndocs.tk/index.html>

## 入门案例分析

在官网手册中，找到这么一段快速入门案例：



可以看出，玩这个东西基本分成4步骤：

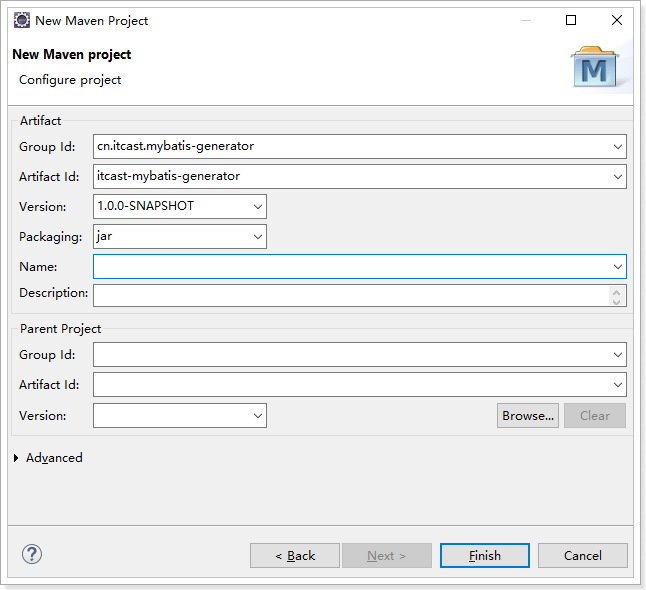
* + 1. 编写MBG的配置文件，默认是叫：generatorConfig.xml
    2. 执行MBG的API，加载配置文件
    3. 生成POJO、Mapper接口、映射文件
    4. 修改mybatis配置，使用这些文件

## MyBatis Generator的使用

接下来，我们就按照前面的分析步骤，来使用下MBG。

### 新建项目，导入依赖

1）新建项目：



1. 导入依赖

注意：MBG需要依赖于JDK1.6以上，因此我们这里指定编译插件为JDK1.7

<dependencies>

<!-- MBG核心 -->

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-core</artifactId>

<version>1.3.4</version>

</dependency>

<!-- MyBatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.2.8</version>

</dependency>

<!-- 数据库驱动 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.34</version>

</dependency>

<!-- 日志相关 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.7</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

</plugins>

</build>

### 编写MBG的配置文件

关于这个配置文件的详解，官方手册的介绍并不详细，因此大家可以参考这篇文章：

<http://blog.csdn.net/isea533/article/details/42102297>

在这里给出一个最佳实践的案例：

<?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:运行环境，包含大部分的配置信息

id：这个运行环境的唯一标示

targetRuntime：配置生成代码风格。默认就是MyBatis3，还可以设置MyBatis3Simple，不会生成复杂条件的查询

-->

<context id=*"testContext"* targetRuntime=*"MyBatis3"* defaultModelType=*"flat"*>

<commentGenerator>

<!-- 是否去除自动生成的注释 true：是 ： false:否 -->

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

<!-- 是否去除日期注释 -->

<property name=*"suppressDate"* 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/mydb"*

userId=*"root"*

password=*"123"*>

</jdbcConnection>

<!-- 默认false,会根据数据精度来判断转换为Java什么类型。推荐

为 true时会强制把数据库Decimal、Numeric数据类型解析为java.math.BigDecimal

-->

<javaTypeResolver>

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

</javaTypeResolver>

<!-- javaModelGenerator:有关实体类的配置

targetPackage:生成实体类所在包

targetProject:生成POJO类的位置

-->

<javaModelGenerator targetPackage=*"cn.itcast.mybatis.pojo"* targetProject=*"src/main/java"*>

<!-- 是否根据表schema生成子包 -->

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

<!-- 是否清理 从数据库返回的值前后的空格 -->

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

</javaModelGenerator>

<!-- sqlMapGenerator：有关映射文件的配置，属性跟上面的javaModelGenerator类似

targetPackage：指定mapper.xml所在包名

targetProject：指定在项目中存放的位置，因为是maven工程，所以放在resource下。

最终Mapper.xml文件的目录就是 src/main/resources下的mappers目录下

-->

<sqlMapGenerator targetPackage=*"mappers"* targetProject=*"src/main/resources"*>

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

</sqlMapGenerator>

<!-- javaClientGenerator：有关Mapper接口的配置，属性跟上面的javaModelGenerator类似

type="XMLMAPPER"：设置以XML格式生成映射而非注解

-->

<javaClientGenerator type=*"XMLMAPPER"* targetPackage=*"cn.itcast.mybatis.mapper"*

targetProject=*"src/main/java"*>

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

</javaClientGenerator>

<!-- 指定数据库表

tableName：表的名称

domainObjectName：指定表对应生成实体类的名称

-->

<table tableName=*"tb\_user"* domainObjectName=*"User"*>

<!-- 设置自增ID的回显功能 -->

<generatedKey column=*"id"* sqlStatement=*"Mysql"*/>

</table>

</context>

</generatorConfiguration>

### 运行MyBatis Generator



我们这里采用Java程序，调用MBG的API来运行。

/\*

\* MyBatis Generator的测试案例

\*/

**public** **class** GeneratorTest {

**public** **static** **void** main(String[] args) **throws** Exception {

List<String> warnings = **new** ArrayList<String>();

// 是否覆盖以前生成的重复代码

**boolean** overwrite = **true**;

// 配置文件的资源路径，会去classpath下找。maven项目中，就是resource下

InputStream in = Resources.getResourceAsStream("generatorConfig.xml");

// 配置解析器

ConfigurationParser cp = **new** ConfigurationParser(warnings);

// 加载配置文件，并解析

Configuration config = cp.parseConfiguration(in);

DefaultShellCallback callback = **new** DefaultShellCallback(overwrite);

// 获取MBG对象

MyBatisGenerator myBatisGenerator = **new** MyBatisGenerator(config, callback, warnings);

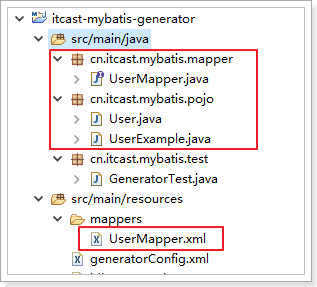
// 生成代码

myBatisGenerator.generate(**null**);

}

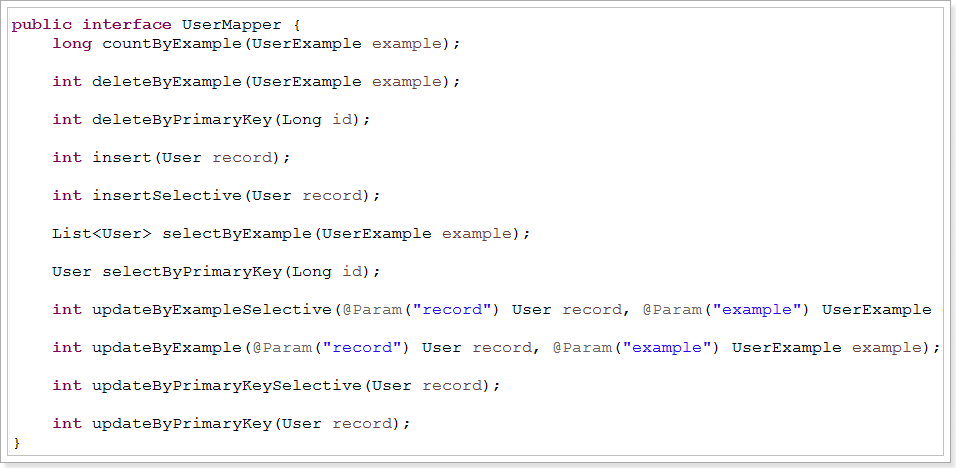
}

### 查看生成的代码：

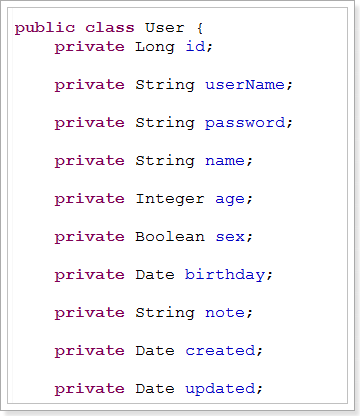


其中：

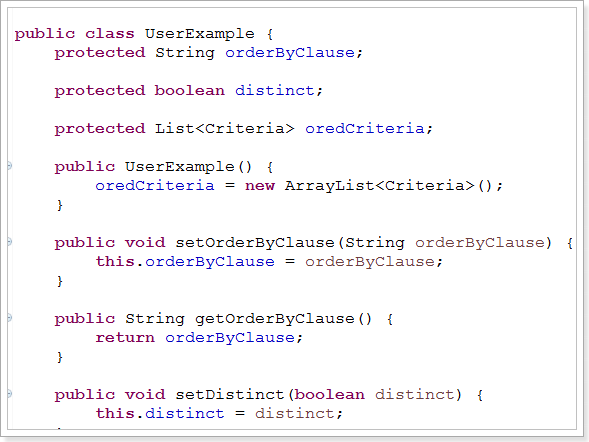
UserMapper：是Mapper接口



User：是实体类



UserExample：是查询User的条件对象：



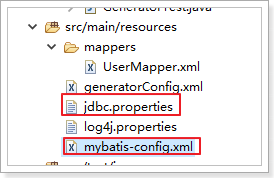
UserMapper.xml：SQL映射文件



### 测试生成的代码

一般情况下，我们会把生成的代码，拿到自己的项目中去使用，这里是测试，所以我们就在这个项目中测试一下就好了。

引入mybatis的全局配置文件和jdbc属性文件：



全局配置：

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

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 引入外部文件,resource：默认就会去项目下的resource目录寻找 -->

<properties resource=*"jdbc.properties"* />

<settings>

<!-- 开启驼峰自动匹配 -->

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"*/>

</settings>

<!-- 定义类型别名，简化XML配置 -->

<typeAliases>

<package name=*"cn.itcast.mybatis.pojo"*/>

</typeAliases>

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<!-- 数据库连接参数 -->

<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>

<!-- 指定映射文件 -->

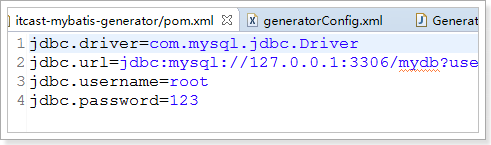
<mappers>

<mapper resource=*"mappers/UserMapper.xml"* />

</mappers>

</configuration>

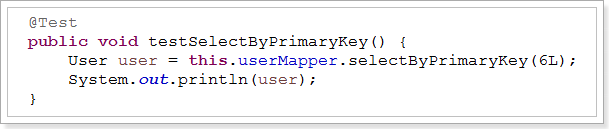
jdbc属性：



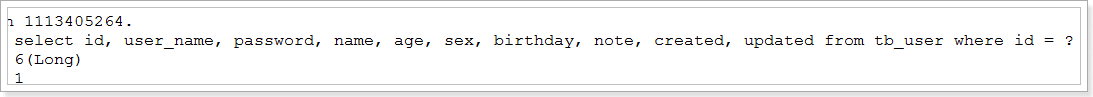
编写测试类：

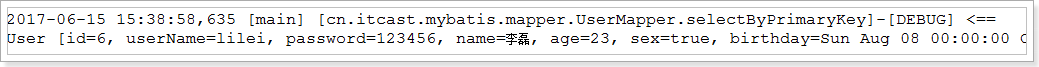


简单查询

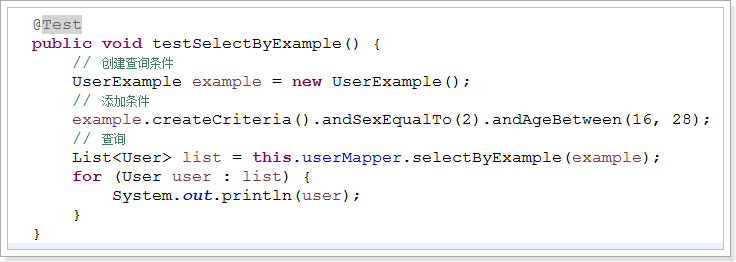


结果：





复杂查询：



结果：

