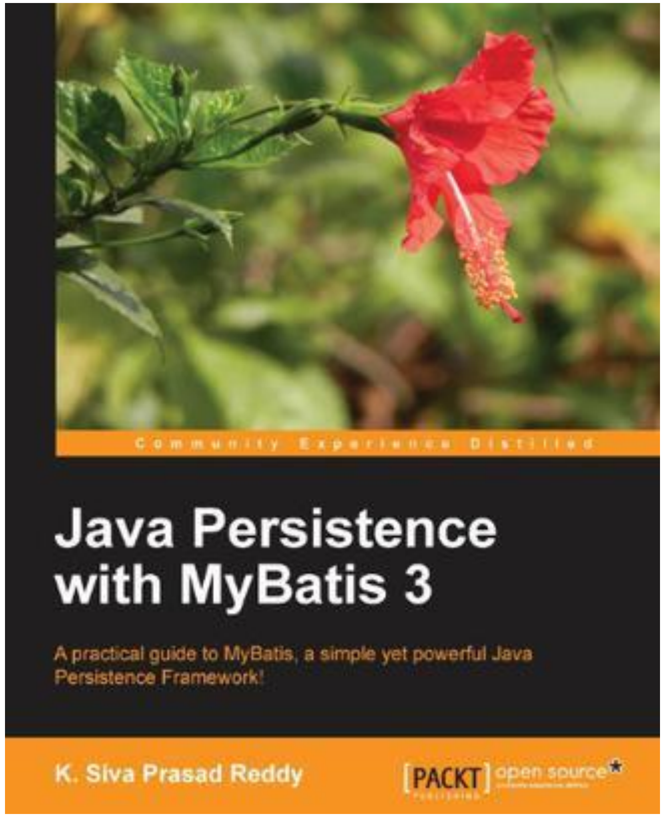
MyBatis 3.x

凯盛软件

推荐书籍


Community Experience Distilled
Java Persistence with MyBatis 3
A practical guide to MyBatis, a simple yet powerful Java Persistence Framework!
K. Siva Prasad Reddy
[PACKT] open source

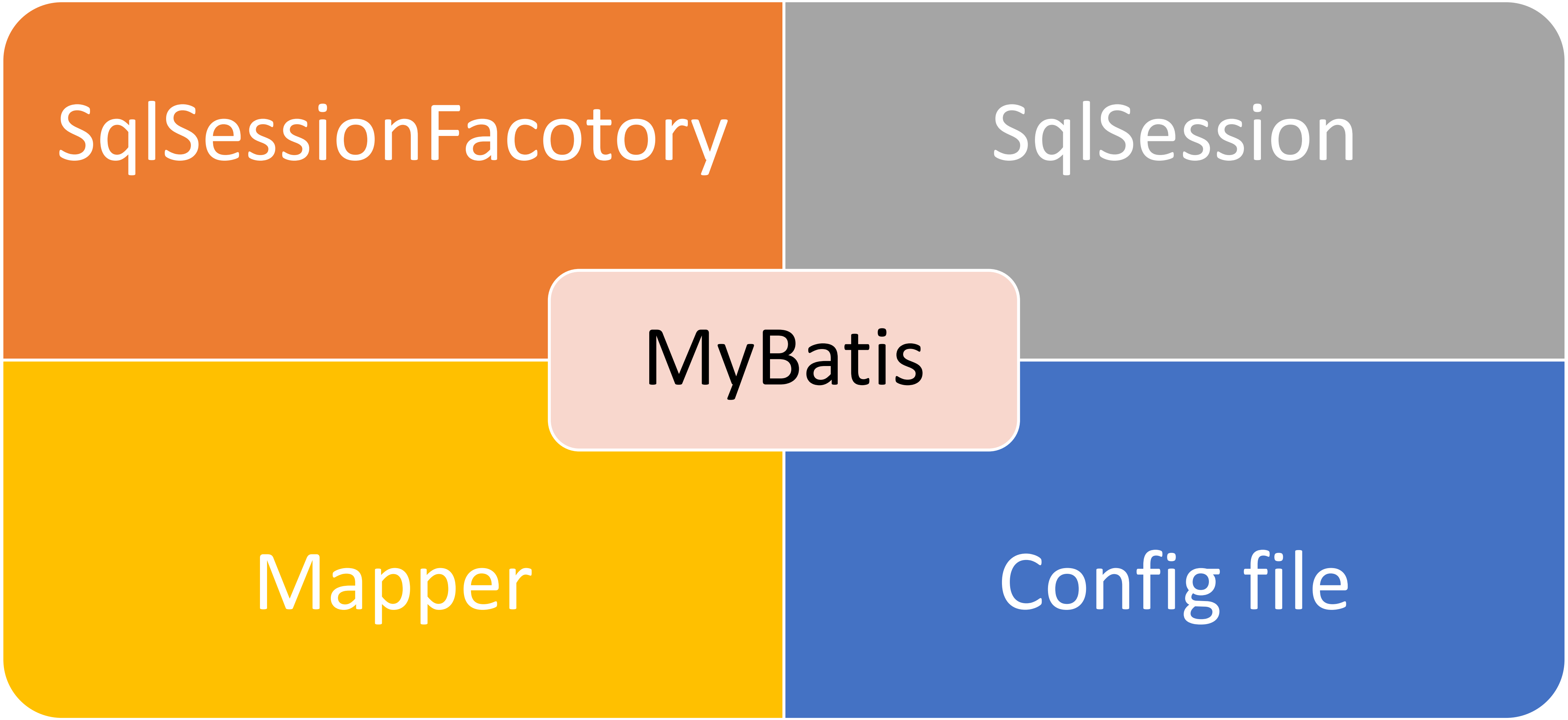
网站

- 官方网址
  - http://www.mybatis.org/mybatis-3/
- GitHub
  - https://github.com/mybatis/

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(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。

MyBatis核心



SqlSessionFacotory

SqlSession

MyBatis

Mapper

Config file

添加Maven依赖

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.x </version>
</dependency>
```

MyBatis配置文件

```
<?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>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"></transactionManager>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql:///mydb"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>

    <mappers>
         <mapper resource=“mapper/usermapper.xml"/>
    </mappers>
</configuration>
```

Mapper文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.kaishengit.mapper.UserMapper">

</mapper>
```

创建SqlSessionFactory对象

```
//从classpath中读取mybatis.xml配置文件
Reader reader = Resources.getResourceAsReader("mybatis.xml");
//根据SqlSessionFactoryBuilder对象构建SqlSessionFactory
SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(reader);
```

创建SqlSession对象

```
//根据SqlSessionFactory对象创建SqlSession对象
SqlSession sqlSession = sessionFactory.openSession();

//code...

sqlSession.close();
```

UserMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.kaishengit.mapper.UserMapper">

    <select id="findById" parameterType="int" resultType="User">
        SELECT * FROM t_user WHERE id = #{id}
    </select>
```

- namespace：命名空间，可以和同样完全限定名中的一个Mapper类对应；
- id：该查询语句的唯一标示
- parameterType：参数的类型
- resultType：返回值类型，如果是自己定义的Entity，需要在config文件中配置别名。

常见数据类型映射

别名	映射的类型
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long

short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

注册别名

- mybatis.xml

```xml
<?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>

    <!--注册别名-->
    <typeAliases>
        <typeAlias type="com.kaishengit.pojo.User" alias="User"/>
    </typeAliases>
```

查询示例

```
//根据SqlSessionFactory对象创建SqlSession对象
SqlSession sqlSession = sessionFactory.openSession();

User user = sqlSession.selectOne("com.kaishengit.mapper.UserMapper.findById",10);
logger.debug("{}",user);

//释放资源
sqlSession.close();
```

insert

```
<insert id="save" parameterType="com.kaishengit.pojo.User" >
    INSERT INTO t_user
(username, password, email, avatar, createtime, loginip, logintime, state)
    VALUES
(#{username},#{password},#{email},#{avatar},#{createtime},#{loginip},#{logintime},#{state})
</insert>
```

```
User user = new User();
user.setUsername("James");
user.setPassword("123123");
user.setState("正常");
user.setEmail("james@google.com");
user.setCreatetime("2016-06-27 12:23:34");

sqlSession.insert("com.kaishengit.mapper.UserMapper.save",user);

sqlSession.commit(); //提交事务
sqlSession.close();
```

事务

- Mybatis默认在操作insert、update、delete时不会自动提交事务，需要通过SqlSession对象的commit()方法来提交事务，通过rollback()方法回滚事务
- 调用SqlSessionFactory对象的openSession(true)方法可以获取一个可以自动提交事务的SqlSession对象

update

```
<update id="update" parameterType="User">
    UPDATE t_user
     set
        password = #{password},
         email = #{email},
         avatar=#{avatar},
         loginip=#{loginip},
         logintime=#{logintime},
         state=#{state}
    where id = #{id}
</update>
```

```
User user = sqlSession.selectOne("com.kaishengit.mapper.UserMapper.findById",13);
user.setLoginip("8.8.8.8");
user.setLogintime("2016-06-14 12:45:44");

sqlSession.update("com.kaishengit.mapper.UserMapper.update",user);
```

delete

```
<delete id="del" parameterType="int">
    DELETE FROM t_user WHERE id = #{id}
</delete>
```

```
sqlSession.delete("com.kaishengit.mapper.UserMapper.del",14);
```

findAll

```
<select id="findAll" resultType="User">
    SELECT * FROM t_user
</select>
```

```
List<User> userList = sqlSession.selectList("com.kaishengit.mapper.UserMapper.findAll");
for(User user : userList) {
    logger.debug("{}",user);
}
sqlSession.close();
```

使用Mapper接口

```java
package com.kaishengit.mapper;
public interface UserMapper {
    User findById(Integer id);
    void save(User user);
    void update(User user);
    void del(Integer id);
    List<User> findAll();
}
```

```java
UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

User user = userMapper.findById(10);
logger.debug("{}",user);

sqlSession.close();
```

动态代理模式 (dynamic proxy)

resultMap

- OneToMany

```
<select id="findById" parameterType="int" resultType="User" resultMap="userMap">
    SELECT t_user.*,t_tag.tagname,t_tag.id as 'tagid'
    FROM t_user
    INNER JOIN t_tag ON t_user.id = t_tag.userid
    WHERE t_user.id = #{id}
</select>


<resultMap id="userMap" type="com.kaishengit.pojo.User">
    <id column="id" property="id"/>
    <result column="username" property="username"/>
    <result column="password" property="password"/>
    <collection property="tagList" ofType="com.kaishengit.pojo.Tag">
        <id column="tagid" property="id"/>
        <result column="tagname" property="tagname"/>
    </collection>
</resultMap>
```

- ManyToOne

```
<select id="findById" parameterType="int" resultType="Topic" resultMap="topicMap">
    SELECT t_topic.*,t_user.username,t_user.avatar,t_node.nodename FROM t_topic
    INNER join t_user on t_topic.userid = t_user.id
    INNER join t_node on t_topic.nodeid = t_node.id
    WHERE  t_topic.id = #{id}
</select>


<resultMap id="topicMap" type="com.kaishengit.pojo.Topic">
    <id column="id" property="id"/>
    <result column="title" property="title"/>
    <result column="text" property="text"/>
    <association property="user" javaType="com.kaishengit.pojo.User" column="userid">
        <id column="userid" property="id"/>
        <result column="username" property="username"/>
    </association>
    <association property="node" javaType="com.kaishengit.pojo.Node" column="nodeid">
        <id column="nodeid" property="id"/>
        <result column="nodename" property="nodename"/>
    </association>
</resultMap>
```

多个查询参数

- 传入对象
- 传入Map

```
<select id="findByParams" resultType="User">
   SELECT * FROM t_user WHERE username = #{param1} and password = #{param2}
</select>
```

```
User findByParams(@Param("username") String username,@Param("pwd") String password);
<select id="findByParams" resultType="User">
    SELECT * FROM t_user WHERE username = #{username} and password = #{pwd}
</select>
```

动态SQL

- if
- choose(when,otherwise)
- trim(where,set)
- foreach

if

```
<select id="findByQueryParam" parameterType="map" resultType="User">
    SELECT * FROM t_user
    where password = #{password}
    <if test="email != null and email != ''">
        and email = #{email}
    </if>
</select>
```

choose ...when

```
<select id="findByQueryParam" parameterType="map" resultType="User">
    SELECT * FROM t_user
    WHERE password = #{password}
    <choose>
        <when test="email != null and email != ''">
            and email = #{email}
        </when>
        <otherwise>
            and 1 = 1
        </otherwise>
    </choose>
</select>
```

where

```
<select id="findByQueryParam" parameterType="map" resultType="User">
    SELECT * FROM t_user
    WHERE
    <if test="username != null and username != ''">
        username = #{username}
    </if>
    <if test="email != null and email != ''">
        and email = #{email}
    </if>
</select>
```

```
<select id="findByQueryParam" parameterType="map" resultType="User">
    SELECT * FROM t_user
    <where>
        <if test="username != null and username != ''">
            username = #{username}
        </if>
        <if test="email != null and email != ''">
            and email = #{email}
        </if>
    </where>
</select>
```

trim

```
<select id="findByQueryParam" parameterType="map" resultType="User">
    SELECT * FROM t_user
    <trim prefix="where" prefixOverrides="and|or">
        <if test="username != null and username != ''">
            username = #{username}
        </if>
        <choose>
            <when test="password != null and password != ''">
                and password = #{password}
            </when>
            <when test="email != null and email != ''">
                and email = #{email}
            </when>
        </choose>
    </trim>
</select>
```

set

```
<update id="updateAuthorIfNecessary" parameterType="domain.blog.Author">
    update Author
    <set>
        <if test="username != null">username=#{username},</if>
        <if test="password != null">password=#{password},</if>
        <if test="email != null">email=#{email},</if>
        <if test="bio != null">bio=#{bio}</if>
    </set>
    where id=#{id}
</update>
```

foreach

```
<select id="findByIds" parameterType="list" resultType="com.kaishengit.pojo.Node">
    SELECT * FROM t_node WHERE id IN
    <foreach collection="list" item="id" separator="," open="(" close=")">
        #{id}
    </foreach>
</select>
```

批量添加

```
<insert id="batchSave" parameterType="list">
    INSERT  INTO t_node(nodename) VALUES
    <foreach collection="list" item="node" separator=",">
        (#{node.nodename})
    </foreach>
</insert>
```

缓存

```
<mapper namespace="com.kaishengit.mapper.NodeMapper">

    <cache/>
```

作用

- 映射语句文件中所有的select语句将被缓存　　　select语句中设置useCache = "false"将不被缓存
- 映射语句文件中的所有insert、update、delete语句会刷新缓存　语句中设置flushCache = "false"将不刷新缓存
- 缓存会使用least recentily used(LRU，最近很少使用的)算法来收回
- 根据时间间隔来刷新缓存，默认不刷新
- 缓存会存储列表集合或对象的1024个引用
- 缓存被视为read/write的缓存

注：pojo类必须是可序列化类

更改默认配置

```
<cache size="2048" readOnly="false" eviction="FIFO" flushInterval="60000"/>
```

- 存储2048个对象
- 不是只读缓存
- 缓存策略为FIFO
- 每隔60秒刷新一次缓存

- eviction回收策略
  - LRU：最近最少使用的，移除长时间不被使用的对象（默认）
  - FIFO：先进先出：按对象进入缓存的顺序来移除他们
  - SOFT：软引用：移除基于垃圾回收器状态和软引用规则的对象
  - WEAK：弱引用：更积极地移除基于垃圾收集器状态和弱引用规则的对象。
- flushInterval（刷新间隔）：可以被设置为任意的正整数
- size（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是 1024。
- readOnly（只读）属性可以被设置为 true 或 false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是false。

MyBatis 注解配置

```
@Insert("insert into dept(deptname) values(#{deptname})")
public void save(Dept dept);


@Delete("delete from dept where id = #{id}")
public void delete(int id);


@Update("update dept set deptname = #{deptname} where id = #{id}")
public void update(Dept dept);
```

mybatis中注解不能完全替代xml，常以注解 + xml或者只使用xml的形式

```
@Select("select * from user where dept_id = #{deptId}")
public List<User> findByDeptId(int deptId);

@Select("select * from user where username = #{name} and password = #{pwd}")
public User findByNameAndPwd(@Param("name")String name,@Param("pwd")String pwd);
```

OneToMany

```
@Select("select * from dept where id = #{id}")
@Results(value = {
        @Result(property = "id",column = "id"),
        @Result(property = "deptname",column = "deptname"),
        @Result(property = "users",javaType = List.class,column ="id", many = @Many(select =
"com.kaishengit.mapper.UserMapper.findByDeptId"))
})
public Dept findById(int id);


@Select("select * from user where dept_id = #{deptId}")
public List<User> findByDeptId(int deptId);
```

ManyToOne

```
@Select("select * from user where id = #{id}")
@Results(value={
        @Result(property = "id",column = "id"),
        @Result(property = "username",column = "username"),
        @Result(property = "password",column = "password"),
        @Result(property = "dept",column = "dept_id",one = @One(select =
"com.kaishengit.mapper.DeptMapper.findById"))
})
public User findById(int id);


@Select("select * from dept where id = #{id}")
public Dept findById(int id);
```

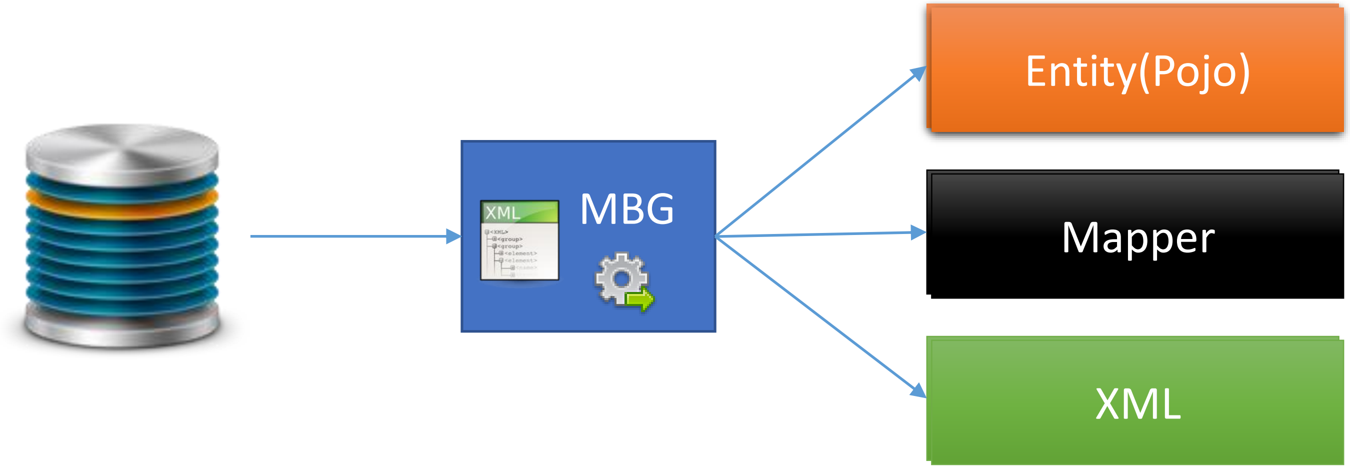
Cache

```
@CacheNamespace
public interface DeptMapper {


@Insert("insert into dept(deptname) values(#{deptname})")
@Options(flushCache = true)
public void save(Dept dept);
```

MyBatis 逆向工程

- http://www.mybatis.org/generator/index.html
- 根据配置文件的规则，自动生成实体类、Mapper接口和XML文件



generatorConfig.xml

https://github.com/fankay/configfiles/blob/master/mybatis/generatorConfig.xml

```
<?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>

    <!-- 连接数据库jar包的位置 -->
    <classPathEntry location="D:\jar\mysql-connector-java-5.1.41.jar"/>

    <!--
    id: 自定义
    targetRuntime : MyBatis3Simple 生成简单的CRUD语句
    MyBatis3 生成带Example的CRUD语句
     -->
    <context id="myConfig" targetRuntime="MyBatis3">

        <!--去掉自动产生的注释-->
        <commentGenerator>
            <!-- 是否去掉自动生成的注释 true是 false 否 -->
            <property name="suppressAllComments" value="true"/>
            <property name="suppressDate" value="true"/>
        </commentGenerator>
        <!-- 连接数据库的信息 -->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
                        connectionURL="jdbc:mysql:///ssm_crm?useSSL=false"
                        userId="root" password="rootroot"/>

        <!-- POJO -->
        <javaModelGenerator targetPackage="com.kaishengit.pojo" targetProject="src/main/java"/>
         <!--XML映射文件-->
        <sqlMapGenerator targetPackage="mapper" targetProject="src/main/resources"/>
         <!--Mapper接口-->
        <javaClientGenerator type="XMLMAPPER"
                             targetPackage="com.kaishengit.mapper"
                             targetProject="src/main/java"/>


        <table tableName="t_user" domainObjectName="User" enableSelectByExample="true"/>
    </context>


</generatorConfiguration>
```

运行

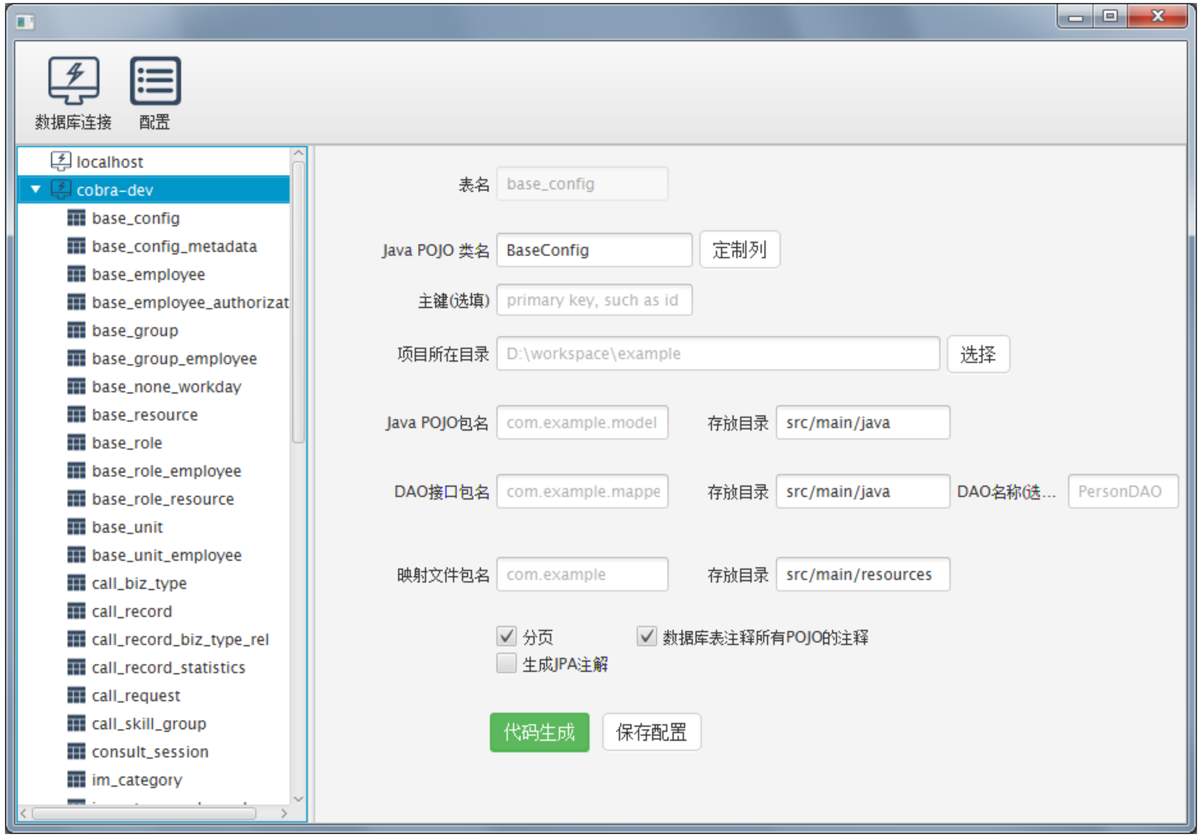
- pom.xml中添加插件

```xml
<plugins>
     <plugin>
          <groupId>org.mybatis.generator</groupId>
          <artifactId>mybatis-generator-maven-plugin</artifactId>
          <version>1.3.5</version>
     </plugin>
</plugins>
```

- mvn命令运行

```
mvn mybatis-generator:generate
```

- gui 方式
  - https://github.com/astarring/mybatis-generator-gui



localhost
cobra-dev
base_config
base_config_metadata
base_employee
base_employee_authorizat
base_group
base_group_employee
base_none_workday
base_resource
base_role
base_role_employee
base_role_resource
base_unit
base_unit_employee
call_biz_type
call_record
call_record_biz_type_rel
call_record_statistics
call_request
call_skill_group
consult_session
im_category

表名 base_config

Java POJO 类名 BaseConfig

主键(选填) primary key, such as id

项目所在目录 D:\workspace\example

Java POJO包名 com.example.model 存放目录 src/main/java

DAO接口包名 com.example.mappe 存放目录 src/main/java DAO名称(选... PersonDAO

映射文件包名 com.example 存放目录 src/main/resources

☑ 分页 ☑ 数据库表注释所有POJO的注释
□ 生成JPA注解

Example类的使用

- 查询全部

```
UserMapper userMapper = session.getMapper(UserMapper.class);

UserExample userExample = new UserExample();
List<User> userList = userMapper.selectByExample(userExample);
for(User user : userList) {
         System.out.println(user.getId() + " -> " + user.getUsername());
}
```

- 根据主键查询

```
UserMapper userMapper = session.getMapper(UserMapper.class);

User user = userMapper.selectByPrimaryKey(1);
System.out.println(user.getUsername());
```

- 根据条件查询

```
UserExample userExample = new UserExample();
userExample.createCriteria().andUsernameEqualTo("admin");

List<User> userList = userMapper.selectByExample(userExample);



UserExample userExample = new UserExample();
userExample.or().andUsernameEqualTo("admin");
userExample.or().andUsernameEqualTo("jack");

List<User> userList = userMapper.selectByExample(userExample);



UserExample userExample = new UserExample();
userExample.createCriteria().andTelEqualTo("138").andStateEqualTo("正常");

List<User> userList = userMapper.selectByExample(userExample);
```

- 排序

```
UserExample userExample = new UserExample();
userExample.createCriteria().andStateEqualTo("正常");
userExample.setOrderByClause("id desc");

List<User> userList = userMapper.selectByExample(userExample);
```

分页插件

- https://github.com/pagehelper/Mybatis-PageHelper
- 使用方法 https://github.com/pagehelper/Mybatis-PageHelper/blob/master/wikis/zh/HowToUse.md
- 添加Maven依赖

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>5.0.0</version>
</dependency>
```

- 配置MyBatis插件（mybatis.xml）

```
<plugins>
    <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>
</plugins>
```

- 使用方法（PageHelper类）

```
//第一页，取5条数据
PageHelper.startPage(1, 5);
UserExample userExample = new UserExample();
List<User> userList = userMapper.selectByExample(userExample);


//从5开始，取10条数据
PageHelper.offsetPage(5, 10);
UserExample userExample = new UserExample();
List<User> userList = userMapper.selectByExample(userExample);

PageHelper.startPage(2, 5);
UserExample userExample = new UserExample();
List<User> userList = userMapper.selectByExample(userExample);
//转换为PageInfo对象
PageInfo<User> pageInfo = new PageInfo<User>(userList);
for(User user : pageInfo.getList()) {
    System.out.println(user.getId() + " -> " + user.getUsername());
}
```

- 使用方法（指定分页参数）

```
@Select("select * from t_user where username = #{userName}")
List<User> findByUserName(@Param("userName") String userName,
                          @Param("pageNum") int pageNum,@Param("pageSize") int pageSize);


List<User> userList = userMapper.findByUserName("admin", 1, 5);
```

```
@Select("select * from t_user where username = #{userName}")
List<User> findByParam(Map<String,Object> param);

Map<String, Object> param = new HashMap<String, Object>();
param.put("userName", "admin");
param.put("pageNum", 1);
param.put("pageSize", 5);

List<User> userList = userMapper.findByParam(param);
```