通用Mapper 3.2.x

http://git.oschina.net/free/Mapper

详细文档请看项目首页

通用Mapper的优势

- 使用通用Mapper可以让你方便的直接使用单表的增删改查方法
- 你不需要使用MyBatis生成器来生成一大堆的XML内容和接口方法。
- 你只需要继承通用Mapper的接口即可获得大量直接可用的方法。
- 表有变化的时候，只需要修改实体类，不需要重新生成接口和XML。

如何使用通用Mapper

1.引入通用Mapper依赖或者Jar包
2.创建自己项目的基础Mapper(Dao)
3.配置通用Mapper
4.MapperScannerConfigurer简单说明
5.修改(或生成)实体类
6.创建实体对应的具体的Dao
7.在Service中使用

1.引入通用Mapper依赖或者Jar包

如果你使用Maven，只需要添加如下依赖：

```
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper</artifactId>
    <!-- 建议使用最新版本,最新版本请从项目首页查找 -->
    <version>x.x.x</version>
</dependency>
```

如果你想引入Jar包，你可以从下面的地址下载：

https://oss.sonatype.org/content/repositories/releases/tk/mybatis/mapper

http://repo1.maven.org/maven2/tk/mybatis/mapper

由于通用Mapper依赖JPA，所以还需要下载persistence-api-1.0.jar：

http://repo1.maven.org/maven2/javax/persistence/persistence-api/1.0/

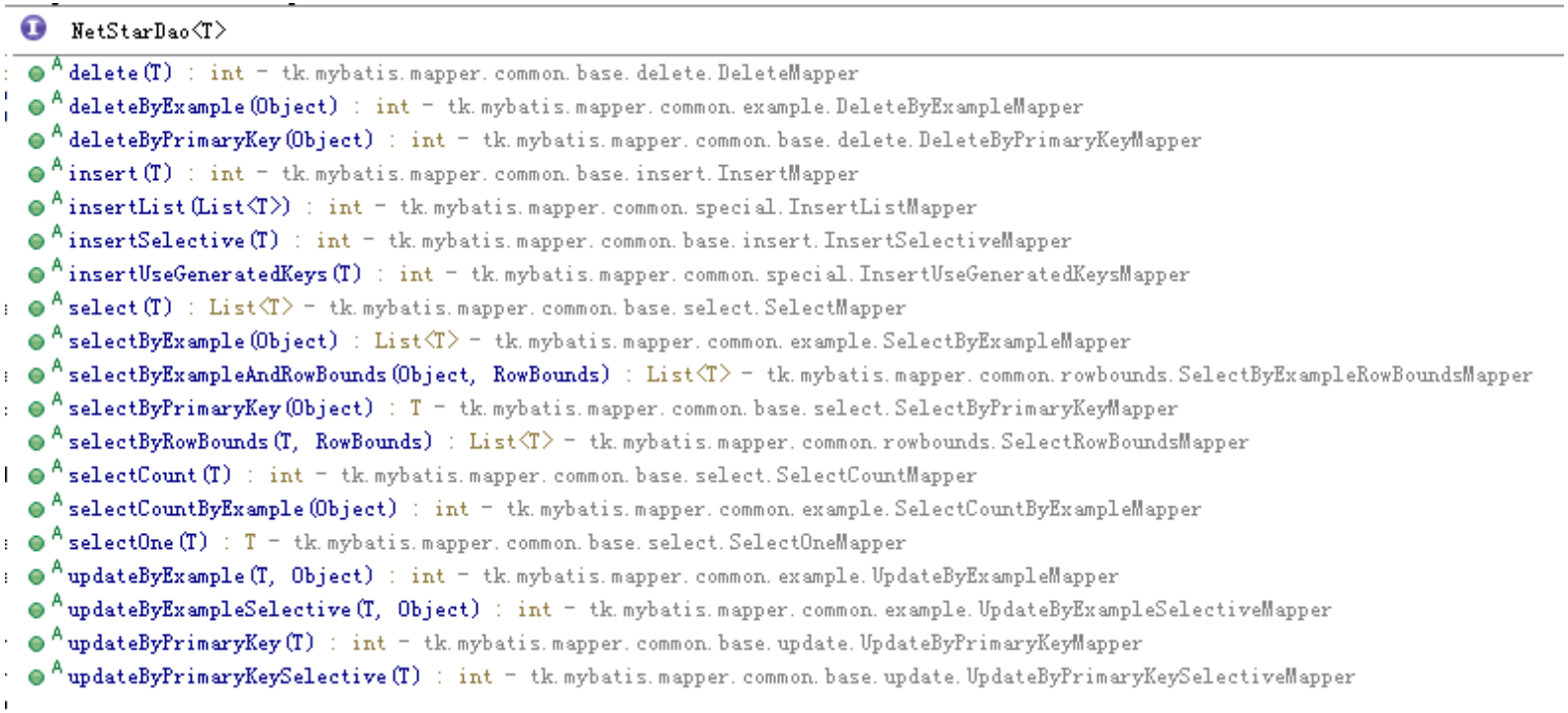
当前最新版本为3.1.2

2.创建自己项目的基础Mapper(Dao)

```java
package com.netstar.util;

import tk.mybatis.mapper.common.Mapper;
import tk.mybatis.mapper.common.MySqlMapper;

/**
 * 通用DAO基础接口，其他的DAO继承该接口即可
 *
 * @author liuzenghui
 *
 * @param <T>
 */
public interface NetStarDao<T> extends Mapper<T>, MySqlMapper<T> {

}

```

这里创建了一个名为NetStarDao<T>的接口，继承了通用Mapper中的Mapper<T>和MySqlMapper<T>接口。

NetStarDao<T>继承后包含的方法



- 在insert和update中带Selective后缀的方法，在insert和update时，不会插入或者更新值为null的列。
- 带Example的方法，支持MyBatis生成器生成的XXXExample和通用Mapper自带的Example(XXX.class)类，Example方法能实现大部分的复杂单表操作，详细的用法可以自己尝试（最后面有个例子）。
- 具体方法的含义可以查看项目文档中的《Mapper3通用接口大全》

3.配置通用Mapper

在MyBatis3.2.x版本以后去掉了MapperInterceptor拦截器。
所以后续版本通过下面的方式进行配置。
下面是原来的扫描接口配置：

```
<!-- 自动扫描Mapper接口生成代理类 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.netstar.**.dao"/>
    <!-- 只有继承了NetStarDao的接口才自动生成代理类，其他的还需要手工编写实现类 -->
    <property name="markerInterface" value="com.netstar.util.NetStarDao"/>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
</bean>
```

在3.2.x版本以后，需要改成下面这样：

```
<!-- 自动扫描Mapper接口生成代理类，这里和上面搜索dao不冲突 -->
<bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.netstar.**.dao"/>
    <!-- 只有继承了NetStarDao的接口才自动生成代理类，其他的还需要手工编写实现类 -->
    <property name="markerInterface" value="com.netstar.util.NetStarDao"/>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
    <property name="properties">
        <value>
        mappers=com.netstar.util.NetStarDao
        INDENTITY=MYSQL
        </value>
    </property>
</bean>
```

主要的变化是将org.mybatis包换成了tk.mybatis包，增加了properties属性进行通用Mapper的配置注入。
注意配置中的com.netstar.util.NetStarDao就是我们前面创建的基础接口

4.MapperScannerConfigurer简单说明

```
<!-- 自动扫描Mapper接口生成代理类，这里和上面搜索dao不冲突 -->
<bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.netstar.**.dao"/>
    <!-- 只有继承了NetStarDao的接口才自动生成代理类，其他的还需要手工编写实现类 -->
    <property name="markerInterface" value="com.netstar.util.NetStarDao"/>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
    <property name="properties">
        <value>
        mappers=com.netstar.util.NetStarDao
        INDENTITY=MYSQL
        </value>
    </property>
</bean>
```

- basePackage可以配置要扫描的包，支持Ant风格的通配符。
- markerInterface可以限制在上述扫描包的基础上，该接口还必须继承这个接口，否则也不会扫描进去。
- sqlSessionFactoryBeanName，一般不需要指定，多数据源时才需要配置，默认值就是sqlSessionFactory
- properties是通用Mapper的属性配置，通过该属性可以配置通用Mapper

5.修改(或生成)实体类

```java
@Table(name = "basfile")
public class BasFile implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(generator = "JDBC")
    private Long id;

    private Long orgid;

    private String filepath;

    private String filetype;

    private Integer filelength;

    private Long userid;

    private Date uploadtime;

    private String originalfilename;

    private String description;
```

对实体增加适当的注解配置（通用Mapper包含了一个可以生成带注解实体类的MyBatis生成器插件，详细信息看项目文档）

6.创建实体对应的Dao

创建BasFile对应的BasFileDao接口，创建完该接口后，该接口就拥有了NetStarDao包含的全部方法。

```java
package com.netstar.cus.bas.dao;

import com.netstar.cus.bas.model.BasFile;
import com.netstar.util.NetStarDao;

public interface BasFileDao extends NetStarDao<BasFile> {

}
```

如果你需要添加自己的接口方法，在这个接口中添加即可。

```java
public interface BasFunsDao extends NetStarDao<BasFuns> {

    List<BasFuns> selectByUserId(long userId);

}
```

另外还要创建一个对应的xml，如下:

```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.netstar.cus.bas.dao.BasFunsDao">
    <!-- 根据用户id获取权限信息 -->
    <select id="selectByUserId" parameterType="long" resultType="com.netstar.cus.bas.model.BasFuns">
        select e.id,a.*
        from BasFuns a
        join BasRoleFuns b on
        a.id=b.funsId
        join BasRole c on c.id=b.roleId
        join BasUserRole d on
        d.roleId=c.id
        join BasUser e on e.id=d.userId
        where
        e.id= #{id}
    </select>
</mapper>
```

需要保证namespace的值是接口的完整名称，另外这个xml也要被扫描到。

```xml
<!-- 系统数据库 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="configLocation" value="classpath:mybatis-config.xml" />
    <property name="mapperLocations" value="classpath*:/com/netstar/**/dao/*.xml" />
    <property name="dataSource" ref="dataSource" />
</bean>
```

实体类注解简单介绍

- @Id 主键字段必须加这个注解，联合主键的时候，可以给多个字段加@Id注解。和PrimaryKey有关的方法都需要该注解标记的字段，所以实体中至少要有一个标记@Id的字段。
- @Table,@Column注解，默认情况下驼峰命名和数据库的下划线命名互相转换，sysUser对应表中的sys_user，当实体类名和字段名不是按照这种默认规则转换时，使用这俩注解指定，例如@Table(name="sysuser")，列类似。
- @Transient注解，当实体中的字段不是表中的字段时，需要加这个注解，这样在和表转换时，会忽略该字段。
- @GeneratedValue(generator="JDBC")注解，这里只介绍这一种，其他的参考Mapper文档，这个配置可以自动回写自增字段的值，相当于XML配置中的userGeneratedKeys="true" keyProperty="id"
- @OrderBy注解可以指定表默认的排序规则
- @NameStyle注解用来配置对象名/字段名和数据库表名/字段之间的转换方式

7.在Service中使用

```java
@Service
public class KeyWordServiceImpl implements KeyWordService {

    @Autowired
    private KeyWordDao keyWordDao;

    /**
     * 获取关键字
     *
     * @param id
     * @return
     */
    @Override
    public KeyWord getKeyWordById(Long id) throws Exception {
        return keyWordDao.selectByPrimaryKey(id);
    }

    /**
     * 分页查询关键字信息
     *
     * @param keyWord
     * @param pageNum
     * @param pageSize
     * @return
     */
    @Override
    public List<KeyWord> getKeyWords(KeyWord keyWord, int pageNum, int pageSize) throws Exception {
        //根据提供的条件判断keyWord对象，然后生成查询条件
        Example example = new Example(KeyWord.class);
        example.createCriteria().andEqualTo("pdfid", keyWord.getPdfid());


        //返回查询结果
        return keyWordDao.selectByExample(example);
    }
```

直接在Service中调用相应的方法即可。

Example方法用法

由于Example方法比较特殊，这里举两个例子：

```
OrgPackagePublishExample example = new OrgPackagePublishExample();
Criteria criteria = example.createCriteria();
//机构ID
if(publish.getOrgid() != null){
    criteria.andOrgidEqualTo(publish.getOrgid());
}
//开始时间
if(publish.getStarttime() != null){
    criteria.andStarttimeGreaterThanOrEqualTo(publish.getStarttime());
}
//结束时间
if(publish.getEndtime() != null){
    criteria.andEndtimeLessThanOrEqualTo(publish.getEndtime());
}
//状态
if(publish.getPackagestate() != null){
    criteria.andPackagestateEqualTo(publish.getPackagestate());
}
return publishDao.selectByExample(example);
```

MyBatis生成器生成的XXXExample

```
Example example = new Example(KeyWord.class);
example.createCriteria()
    .andEqualTo("pdfid", keyWord.getPdfid())
    .andLike("keyword", "%" + keyWord.getKeyword() + "%");
//返回查询结果
return keyWordDao.selectByExample(example);
```

通用Example(XX.class)