1.如果是传统做法。

<bean id="sqlSessionFactory" abstract="true" class="org.mybatis.spring.SqlSessionFactoryBean">  
 <property name="mapperLocations" value="classpath:mybatis/mapper-\*.xml" />  
 <property name="configLocation" value="classpath:mybatis/MyBatisConfiguration.xml" />  
</bean>  
  
<!-- 写 -->  
<bean id="sqlSessionTemplate\_w" class="org.mybatis.spring.SqlSessionTemplate" scope="prototype">  
 <constructor-arg index="0">  
 <bean parent="sqlSessionFactory">  
 <property name="dataSource" ref="dataSource\_w" />  
 </bean>  
 </constructor-arg>  
 <!-- 如果修改为BATCH，无法获取影响的行数 -->  
 <constructor-arg index="1" value="SIMPLE" />  
</bean>  
  
<!-- 读 -->  
<bean id="sqlSessionTemplate\_r" class="org.mybatis.spring.SqlSessionTemplate" scope="prototype">  
 <constructor-arg index="0">  
 <bean parent="sqlSessionFactory">  
 <property name="dataSource" ref="dataSource\_r" />  
 </bean>  
 </constructor-arg>  
 <constructor-arg index="1" value="SIMPLE" />   
</bean>

2.但我们是spring boot，约定大于配置时我们的节操，先不考虑读写分离，我们使用默认注入sqlSessionTemplate实例的方式，并且这个实例是单例的。SqlSessionFactory实例也是默认注入的，一切配置都与我们无关…………

@Repository  
public class AuthClientInfoDaoImpl implements IAuthClientInfoDao{  
  
 private static final String *NAME\_SPACE* = "com.glitter.spring.boot.persistence.dao.IAuthClientInfoDao";  
  
 @Autowired  
 SqlSessionTemplate sqlSessionTemplate;  
  
 @Override  
 public int insert(AuthClientInfo authClientInfo) {  
 return sqlSessionTemplate.insert(*NAME\_SPACE* + ".insert", authClientInfo);  
 }  
  
 @Override  
 public int deleteById(Long id) {  
 return sqlSessionTemplate.delete(*NAME\_SPACE* + ".deleteById", id);  
 }

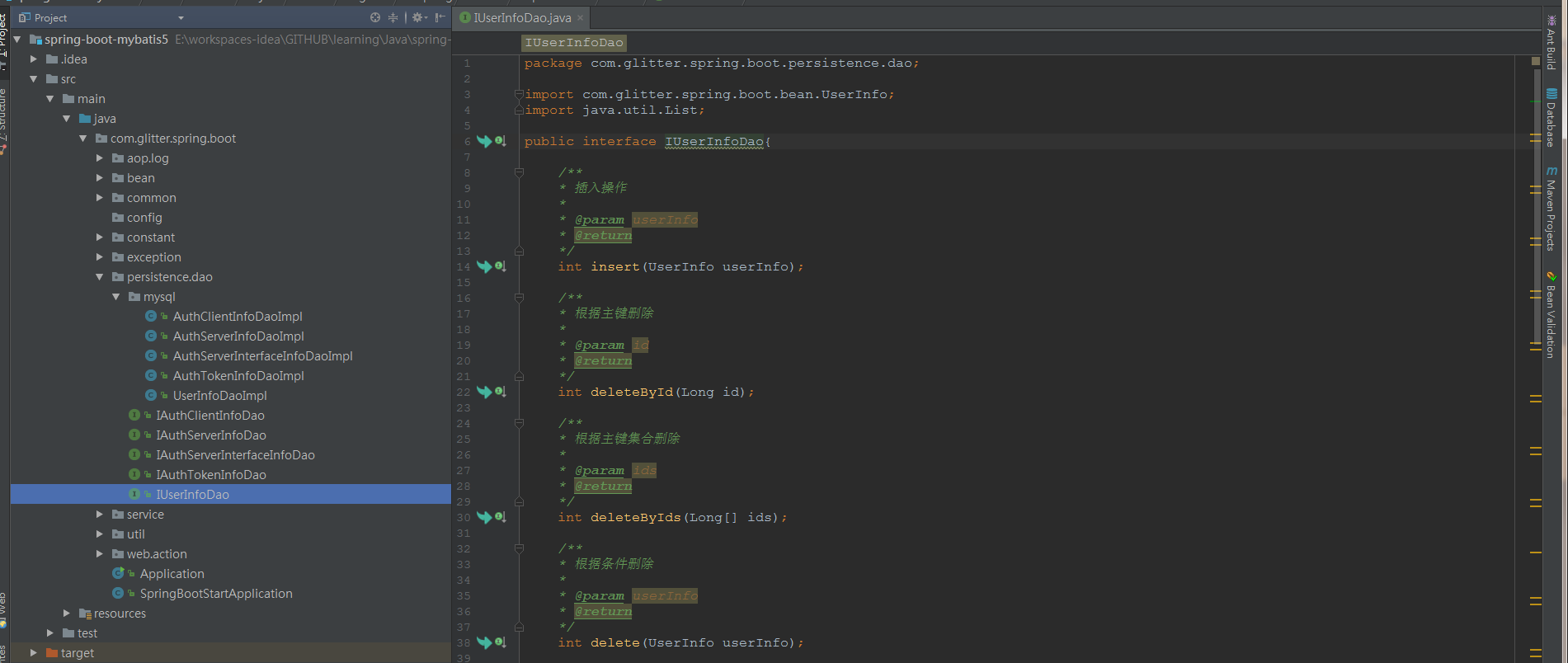
3.application.yml的配置没什么变化。

**spring:** *# 数据源连接池配置,spring开头的配置项都是spring boot框架直接读取使用的* **datasource:  
 driver-class-name:** com.mysql.jdbc.Driver  
 **url:** jdbc:mysql://127.0.0.1:3306/demo\_client  
 **username:** root  
 **password:** root  
 **hikari.maxLifetime:** 1765000  
 **hikari.maximumPoolSize:** 15  
  
**mybatis:** *# mapper.xml文件位置* **mapper-locations:** classpath:mybatis/\*.xml  
 *# 别名配置,看个人习惯,也可以不配置* **type-aliases-package:** com.glitter.spring.boot.bean  
 *# 这里的配置与mybatis-config.xml的settings属性配置时一一对应的,不同点是这里属性名要换成中划线* **configuration:  
 cache-enabled:** false  
 **lazy-loading-enabled:** false  
 **aggressive-lazy-loading:** false  
 **multiple-result-sets-enabled:** true  
 **use-column-label:** true  
 **use-generated-keys:** true  
 **auto-mapping-behavior:** FULL  
 **default-statement-timeout:** 25000  
 **default-fetch-size:** 100  
 **map-underscore-to-camel-case:** true

4.Application类的配置。

@SpringBootApplication  
public class Application {  
  
 private static final Logger logger = LoggerFactory.*getLogger*(Application.class);  
  
 public static void main(String[] args){  
 System.*out*.println("main方法开始");  
 SpringApplication.*run*(Application.class, args);  
 System.*out*.println("main方法结束");  
  
 Long begin = System.*currentTimeMillis*();  
 logger.info("开始时间:" + begin);  
 for (int i = 0; i < 1; i++) {  
 logger.trace("Application.trace...................................."+i);  
 logger.debug("Application.debug...................................."+i);  
 logger.info("Application.info...................................."+i);  
 logger.warn("Application.warn...................................."+i);  
 logger.error("Application.error...................................."+i);  
 }  
 Long end = System.*currentTimeMillis*();  
 logger.info("结束时间:" + end);  
 logger.info("耗时:" + (end - begin) + "毫秒");  
 }  
  
}

5.目录结构。



6.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.glitter.spring.boot.persistence.dao.IUserInfoDao">  
 <resultMap id="BaseResultMap" type="com.glitter.spring.boot.bean.UserInfo">  
 <id column="id" property="id" jdbcType="BIGINT"/>  
 <result column="account" property="account" jdbcType="VARCHAR"/>  
 <result column="password" property="password" jdbcType="VARCHAR"/>  
 <result column="phone" property="phone" jdbcType="VARCHAR"/>  
 <result column="email" property="email" jdbcType="VARCHAR"/>  
 <result column="phone\_verified" property="phoneVerified" jdbcType="BIT"/>  
 <result column="email\_verified" property="emailVerified" jdbcType="BIT"/>  
 <result column="full\_name" property="fullName" jdbcType="VARCHAR"/>  
 <result column="nick\_name" property="nickName" jdbcType="VARCHAR"/>  
 <result column="age" property="age" jdbcType="TINYINT"/>  
 <result column="height" property="height" jdbcType="SMALLINT"/>  
 <result column="remark" property="remark" jdbcType="VARCHAR"/>  
 <result column="delete\_flag" property="deleteFlag" jdbcType="BIT"/>  
 <result column="register\_time" property="registerTime" jdbcType="TIMESTAMP"/>  
 <result column="create\_time" property="createTime" jdbcType="TIMESTAMP"/>  
 <result column="update\_time" property="updateTime" jdbcType="TIMESTAMP"/>  
 </resultMap>  
  
 <sql id="baseColumns">  
 id  
 ,account  
 ,phone  
 ,email  
 ,delete\_flag  
 ,create\_time  
 ,update\_time  
 </sql>  
  
 <sql id="whereSql">  
 <if test="id != null">  
 and id=#{id}  
 </if>  
 <if test="account != null ">  
 and account=#{account}  
 </if>  
 <if test="password != null ">  
 and password=#{password}  
 </if>  
 ……………………………………….

</sql>

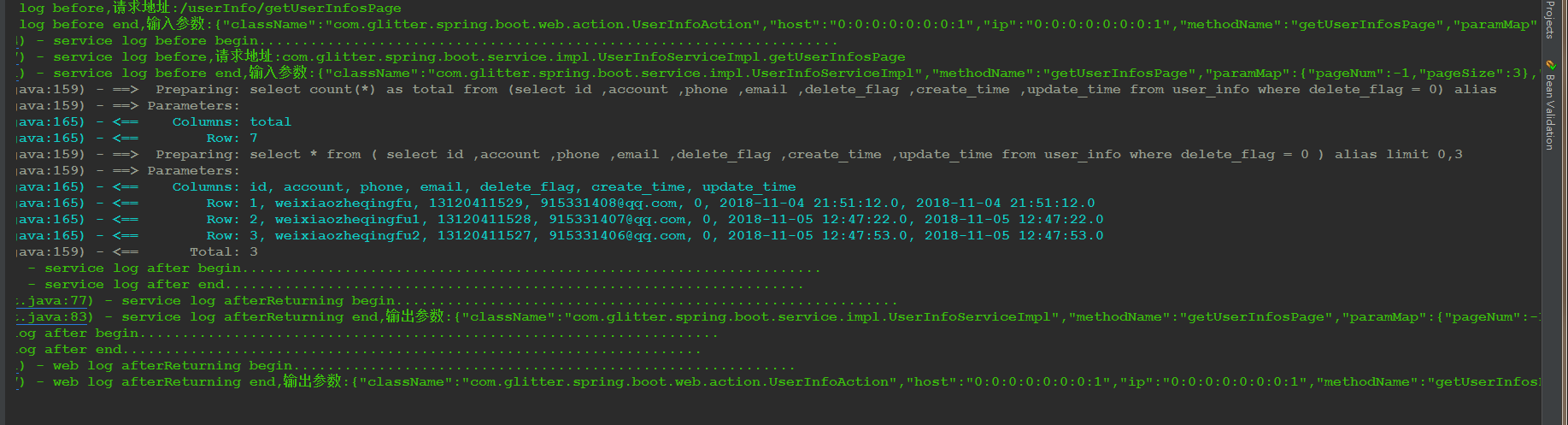
……………………………………….

<delete id="deleteById">  
 delete from user\_info where id =#{id}  
</delete>

……………………………………….

</mapper>

7.还是那个分页接口，运行效果如下：



[8.注意Application中的@MapperScan和com.glitter.spring.boot.persistence.dao.mysql](mailto:8.注意Application中的@MapperScan和com.glitter.spring.boot.persistence.dao.mysql)下的实现类不能同时出现，否则dao层接口相当于有两个实现类了，mybatis和spring boot也懵逼了，不知道该使用哪个了。

也就是说喜闻乐见的dao接口直接映射mapper.xml的方式和使用功能强大灵活的sqlSessionTemplate对象模板的方式不能同时使用。推荐使用sqlSessionTemplate对象模板的方式。

@SpringBootApplication  
@MapperScan({"com.glitter.spring.boot.persistence.dao"})  
public class Application {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(Application.class);  
  
 public static void main(String[] args){  
  
 }  
  
}

