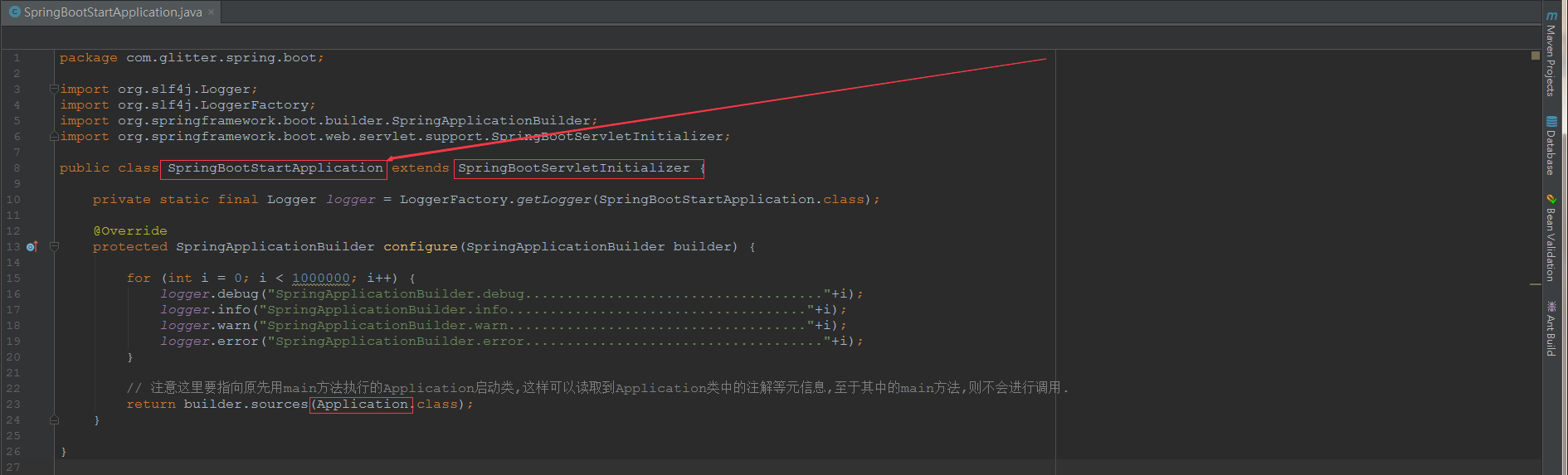
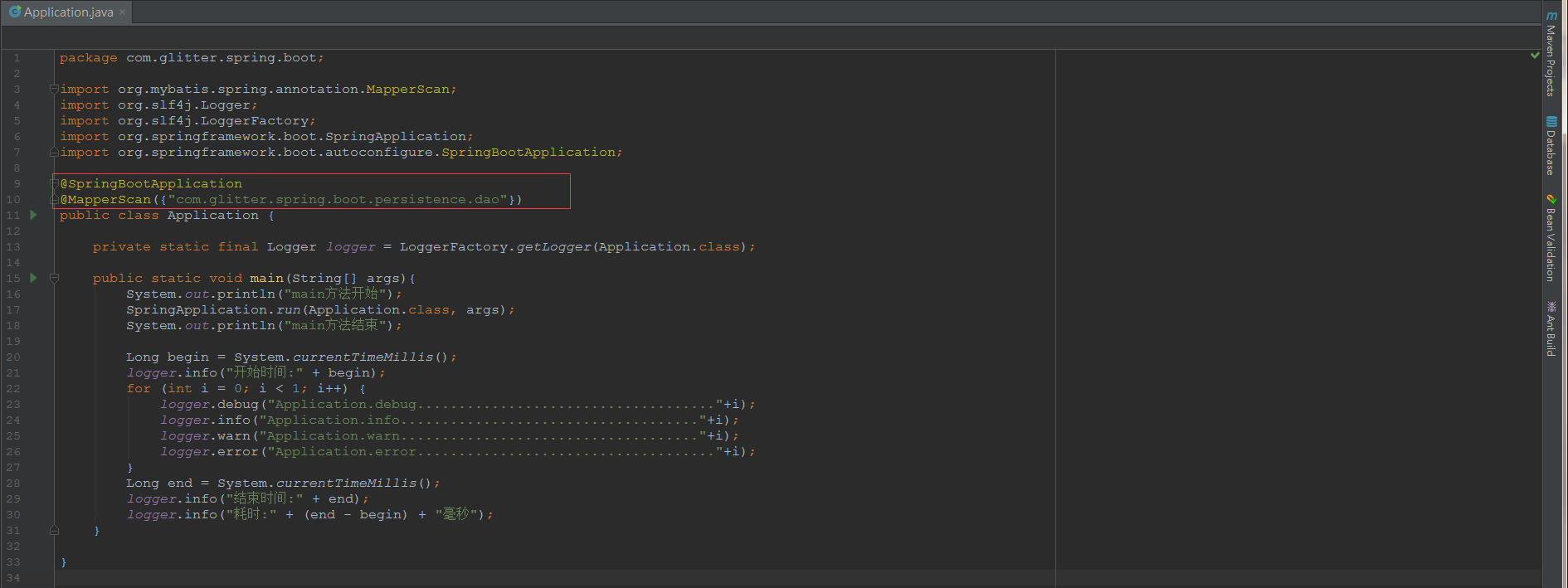
1.spring boot项目，在tomcat容器启动时，（tomcat8.0以上，支持servlet3.0api），springboot相关jar包利用servlet3.0支持spi机制，进行容器启动时的springboot初始化，初始化过程中会读取application.xml或application.yml文件的配置信息，

同时会把所有的配置过程都划归到注解中去（推荐），当然也支持xml。

springboot同时会加载项目中使用注解配置的类，比如@Coniguration注解，这就相当于是原来的配置文件，如果这个类实现了WebMvcConfigurer接口，就可以在其中配置很多Web特性的东西了，比如filter，servlet，listener，interceptor等，这就大体相当于原来的web.xml中做的事情了。

只是执行时机和执行方式发生了变化，web.xml是tomcat容器读取并进行相关的配置的。而这个类中filter，servlet，listener的是springboot利用servlet3.0支持spi和动态加载servlet、filter的api层面的支持，进行动态生成配置的。至于interceptor那是springboot自己的东西，不用多说了，想怎么玩怎么玩。







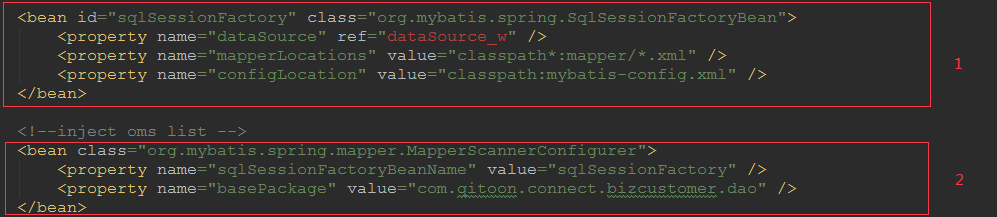
2. pom文件配置mybatis相关jar包依赖。

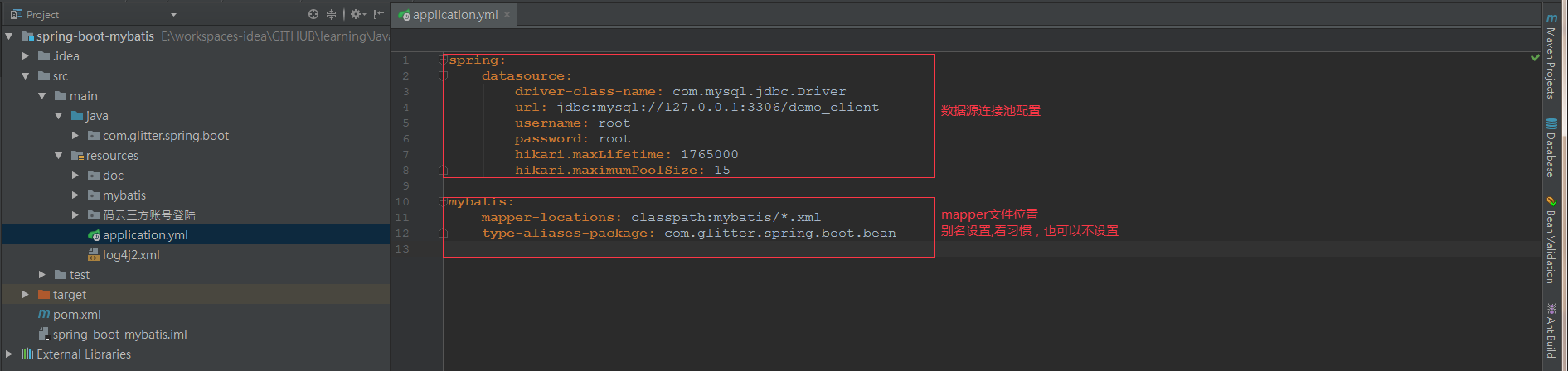
<!-- mybatis -->  
<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>1.3.2</version>  
</dependency>  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.47</version>  
</dependency>  
<!-- 分页插件,可以做一个自己的分页插件 -->  
<dependency>  
 <groupId>com.github.pagehelper</groupId>  
 <artifactId>pagehelper-spring-boot-starter</artifactId>  
 <version>1.2.5</version>  
</dependency>

3.数据源连接池配置和mybaitis的mapper文件路径配置，spring boot默认依赖hikri连接池jar包，并且在application.yml中如果没有特别指明spring.datasource.type类型属性的话，默认配置的就是使用hikari数据库连接池。

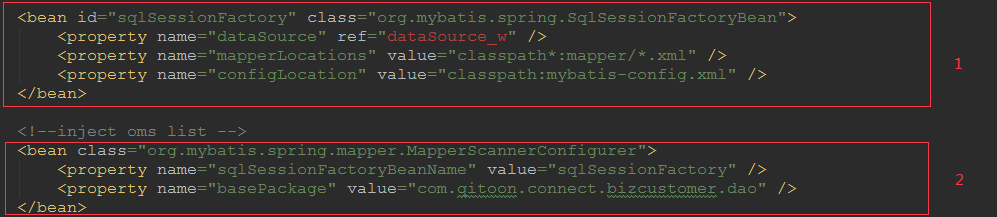
另外如果项目中只配置了一个数据源的话，mybatis的SqlSessionFactoryBean默认也是注入使用该数据源作为获取数据库连接的连接池使用的，并且SqlSessionFactoryBean对象不需要再特别申明就会被默认创建出来，当然也可以指定他要使用哪个数据源创建，不过那是多数据源的事情了。

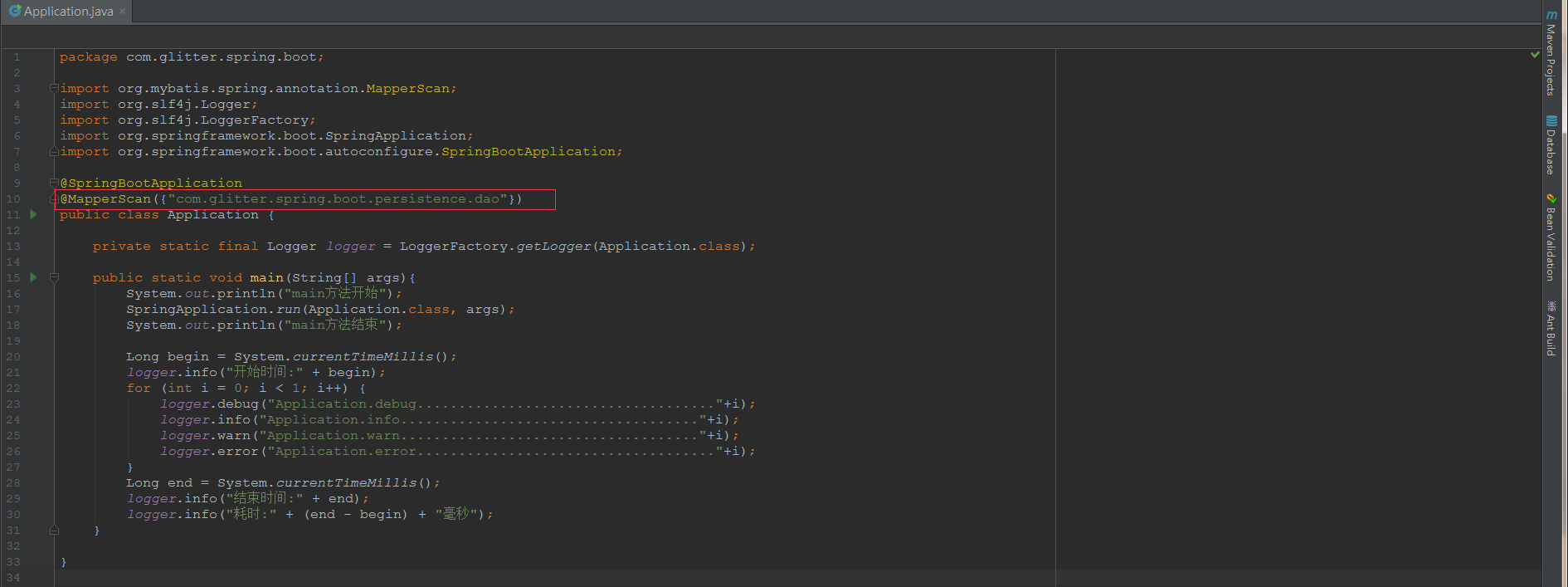
这相当于是传统中的第一步配置。



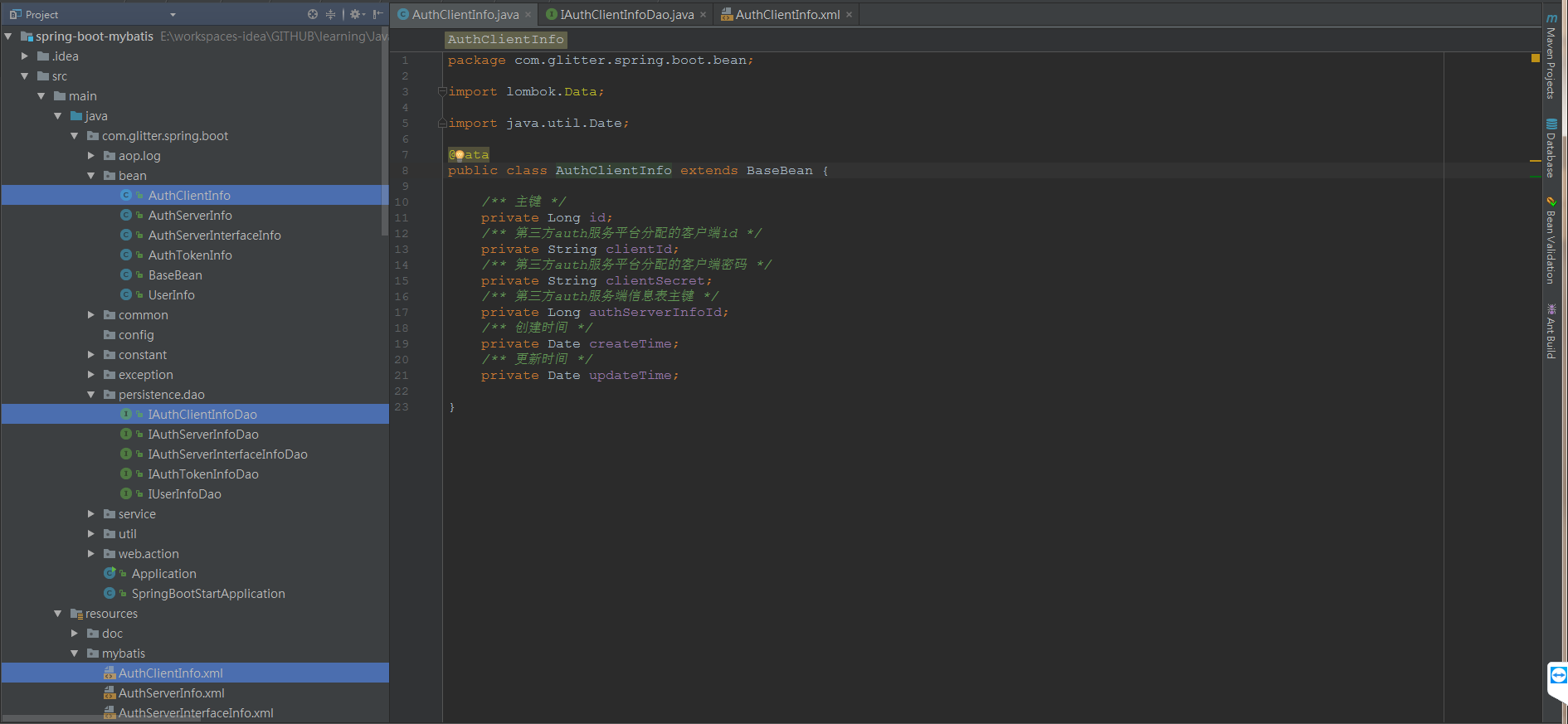


4.指明mybatis扫描的接口路径，背后是创建了MapperScannerConfigurer对象，并且该对象中的sqlSessionFactoryBean就是默认创建的出来的那个bean。相当于传统中的第2步。

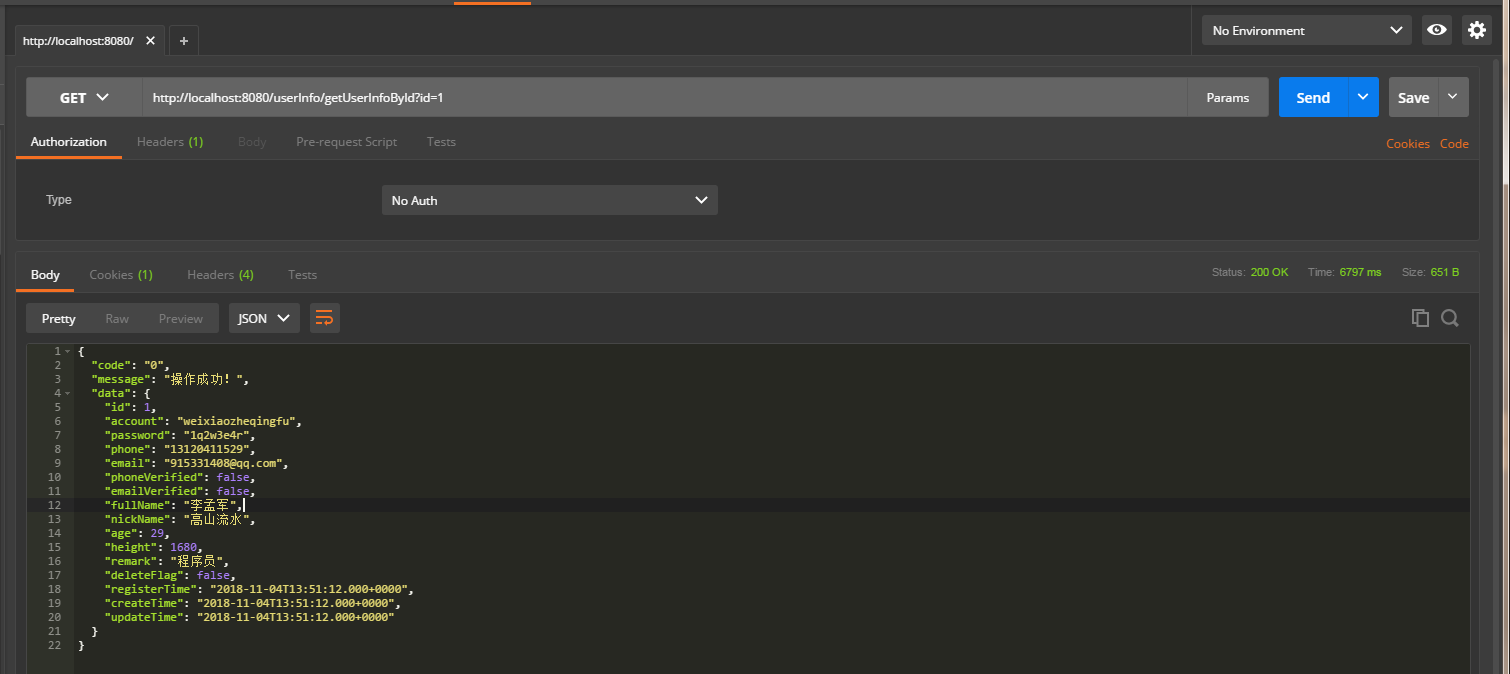




5.新建数据库，并反向生成对应的实体类，dao层接口类，mapper.xml文件。



6.可以进行测试了。



7.后台log4j2打印sql日志信息配置，先明确一点，是否打印sql，其级别控制在于项目本身的日志级别控制，因为mybatis在注入日志的class信息时注入的是项目中dao层接口的class，相当于这个日志是dao接口打出的，那么dao接口所在的项目路径控制的日志输出级别就控制了sql是否打印。

另外mybatis中sql语句、入参、影响行数的日志打印时都是logger.debug(…)； 而查询结果信息使用的是logger.trace(…)进行的打印。

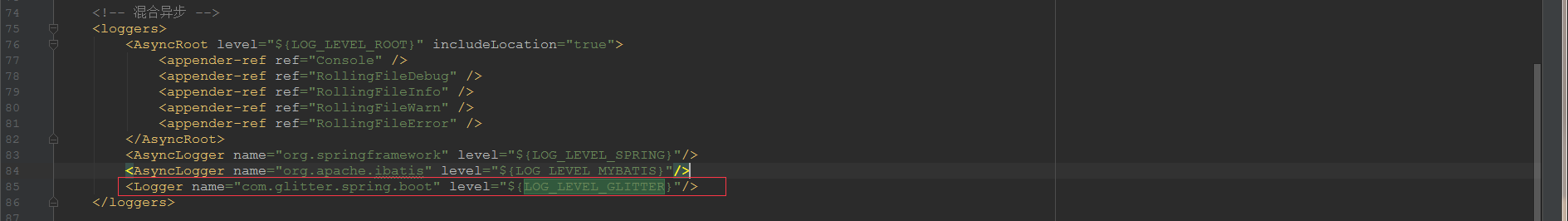
所以如果还想要看返回值，就要将项目的日志级别调整为TRACE级别即可。

对于org.apache.ibatis的配置，控制的是：在mybatis源码中打印日志时，mybatis注入的logger的class是mybatis自己的类时，也就是他确实是已自己的身份在打印自己的日志时，即正常打印自己的日志时，日志的输出情况。一般我们不需要关注这些日志。我们需要关注的往往是sql出入参。

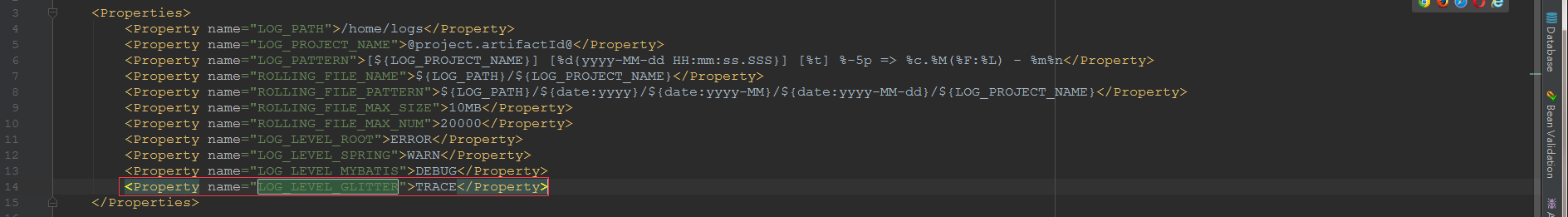
而sql的出入参的日志打印并不是这里控制的，mybatis交给了项目源码自己的日志级别来控制了。

<AsyncLogger name="org.apache.ibatis" level="${LOG\_LEVEL\_MYBATIS}"/>

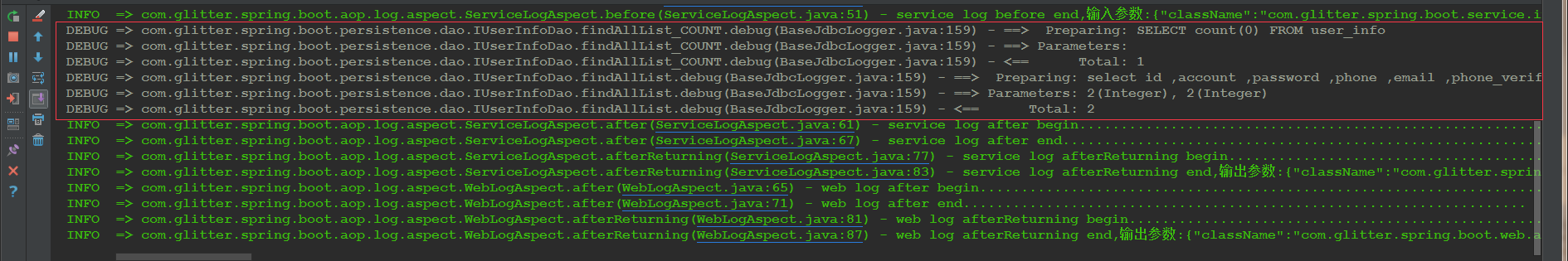
<Property name="LOG\_LEVEL\_MYBATIS">DEBUG</Property>



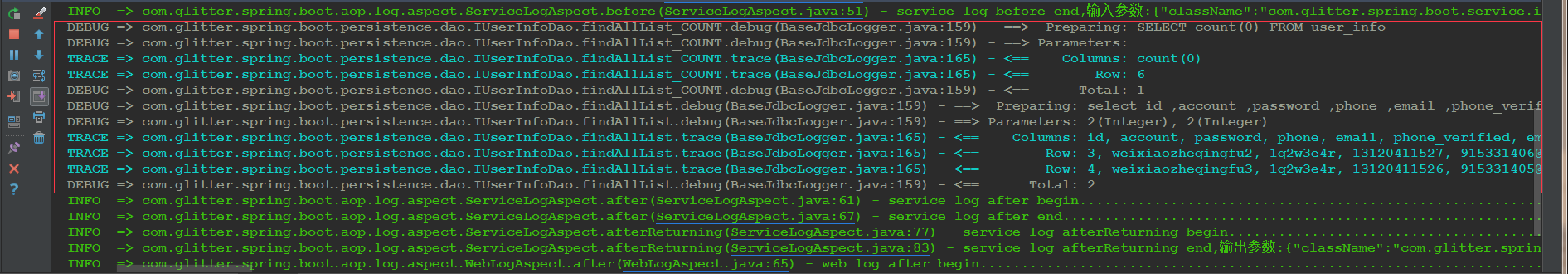
控制台如果对级别有过滤，要保证过滤后能打印TRACE或DEBUG级别及以上才行，如果没有配置过滤，那就算了，这样sql脚本就可以在控制台输出了。



项目日志级别是DEBUG级别的日志效果。



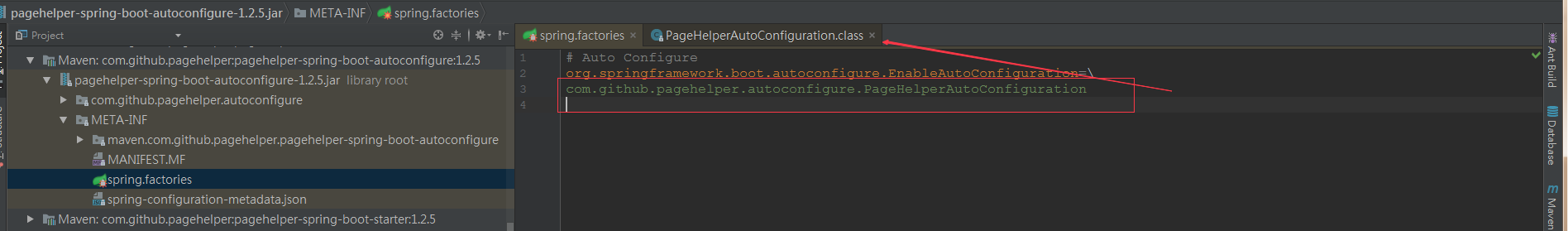
项目日志级别是TRACE级别的日志效果。



8.分页插件的使用，可以发现，虽然没有配置指定mybatis的分页插件配置，但是依然可以使用，这其中的运行原理有待考证。没有配置的情况下，分页插件中使用的各种影响到分页最终效果的属性配置值都是使用的默认值，这不一定能满足我们的需要。

所以还是建议通过指定mybatis分页插件配置的形式来配置pagehelper插件，同时指定其分页属性值，使其分页特性适合我们的需要。这将在后面的项目中专门列出。

原理考证了，是通过spring boot的@Configuration注解的方式将插件进行实例化管理。



//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by Fernflower decompiler)  
//  
  
package com.github.pagehelper.autoconfigure;  
  
import com.github.pagehelper.PageInterceptor;  
import com.github.pagehelper.autoconfigure.PageHelperProperties;  
import java.util.Iterator;  
import java.util.List;  
import java.util.Properties;  
import javax.annotation.PostConstruct;  
import org.apache.ibatis.session.SqlSessionFactory;  
import org.mybatis.spring.boot.autoconfigure.MybatisAutoConfiguration;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.autoconfigure.AutoConfigureAfter;  
import org.springframework.boot.autoconfigure.condition.ConditionalOnBean;  
import org.springframework.boot.context.properties.ConfigurationProperties;  
import org.springframework.boot.context.properties.EnableConfigurationProperties;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
@ConditionalOnBean({SqlSessionFactory.class})  
@EnableConfigurationProperties({PageHelperProperties.class})  
@AutoConfigureAfter({MybatisAutoConfiguration.class})  
public class PageHelperAutoConfiguration {  
 @Autowired  
 private List<SqlSessionFactory> sqlSessionFactoryList;  
 @Autowired  
 private PageHelperProperties properties;  
  
 public PageHelperAutoConfiguration() {  
 }  
  
 @Bean  
 @ConfigurationProperties(  
 prefix = "pagehelper"  
 )  
 public Properties pageHelperProperties() {  
 return new Properties();  
 }  
  
 @PostConstruct  
 public void addPageInterceptor() {  
 PageInterceptor interceptor = new PageInterceptor();  
 Properties properties = new Properties();  
 properties.putAll(this.pageHelperProperties());  
 properties.putAll(this.properties.getProperties());  
 interceptor.setProperties(properties);  
 Iterator var3 = this.sqlSessionFactoryList.iterator();  
  
 while(var3.hasNext()) {  
 SqlSessionFactory sqlSessionFactory = (SqlSessionFactory)var3.next();  
 sqlSessionFactory.getConfiguration().addInterceptor(interceptor);  
 }  
  
 }  
}

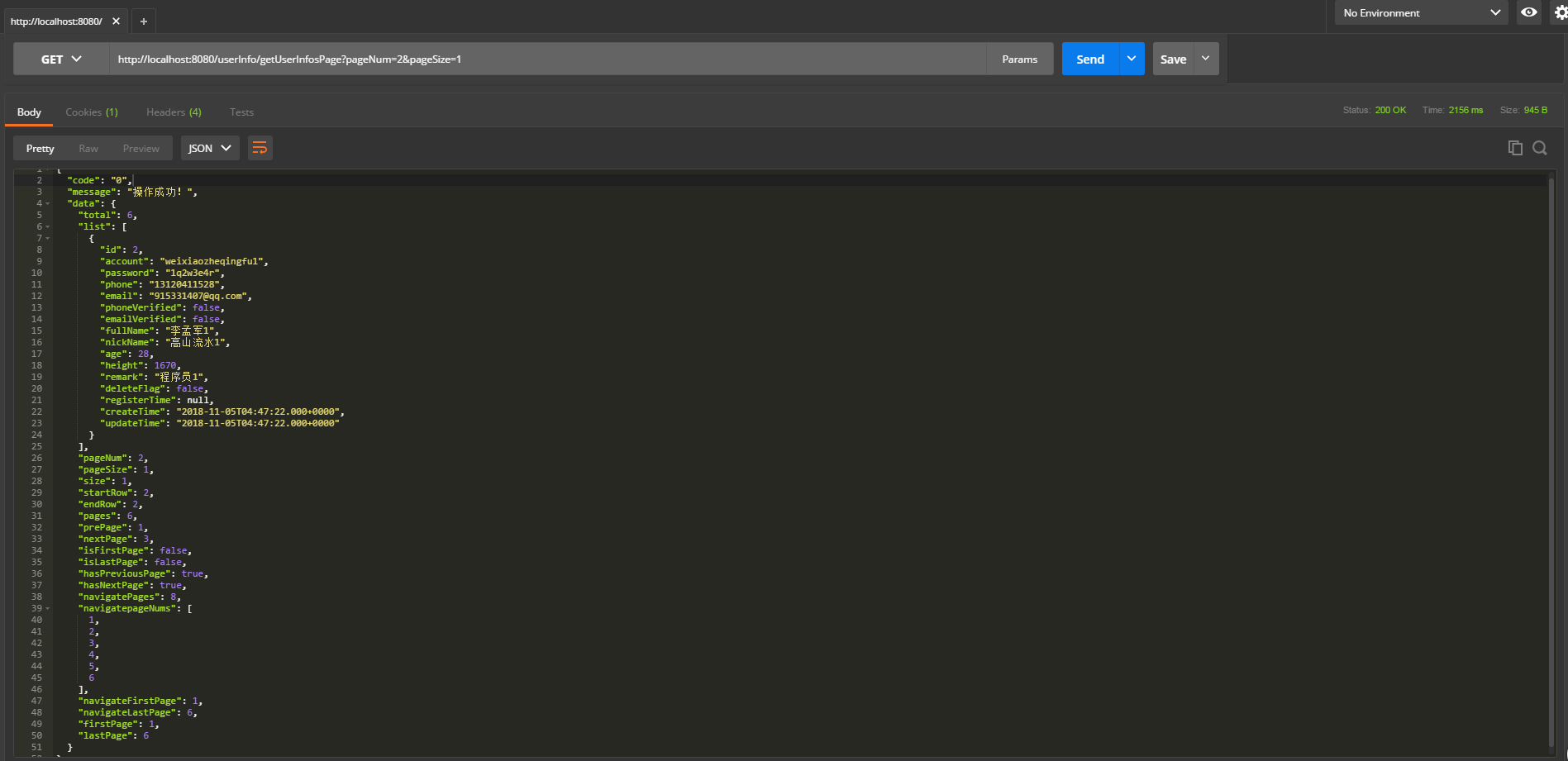
(1). 导入分页插件jar包

<!-- 分页插件,可以做一个自己的分页插件 -->  
<dependency>  
 <groupId>com.github.pagehelper</groupId>  
 <artifactId>pagehelper-spring-boot-starter</artifactId>  
 <version>1.2.5</version>  
</dependency>

(2).代码直接写。

@Override  
public PageInfo<UserInfo> getUserInfosPage(Integer pageNum, Integer pageSize) {  
 PageHelper.*startPage*(pageNum, pageSize);  
 List<UserInfo> userInfos = userInfoDao.findAllList();  
 PageInfo<UserInfo> pageInfo = new PageInfo<>(userInfos);  
 return pageInfo;  
}

(3).执行结果。



9.整个过程相当于传统的“1和2”步的组合，这种组合适用于小项目，因为通常只有一个数据源，且很难做读写数据源的分离。所以后续会继续写“1和3”步的组合在spring boot下是如何进行配置使用的，并且会进一步展开多数据源和读写分离以及mycat的使用。

10.另外，可以不写

@MapperScan({"com.glitter.spring.boot.persistence.dao"})

那就需要在所有的dao接口中写@Mapper注解，也可以，就是比较麻烦点喽

@Mapper  
public interface IUserInfoDao{  
  
 */\*\*  
 \* 插入操作  
 \*  
 \** ***@param*** *userInfo  
 \** ***@return*** *\*/* int insert(UserInfo userInfo);  
  
 */\*\*  
 \* 根据主键删除  
 \*  
 \** ***@param*** *id  
 \** ***@return*** *\*/* int deleteById(Long id);