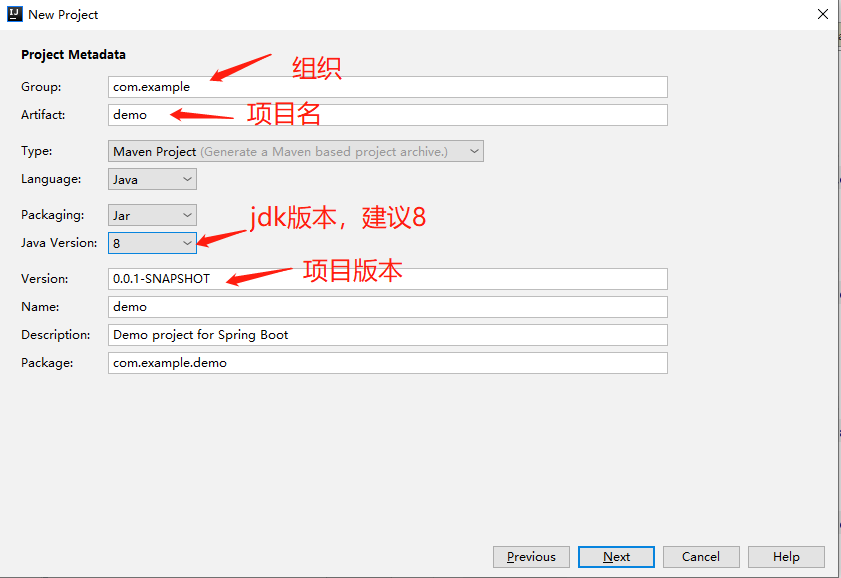
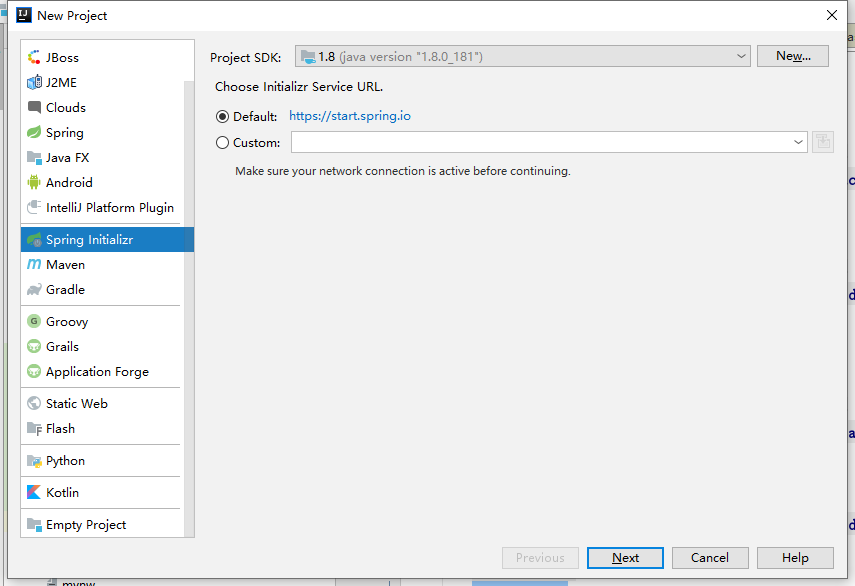
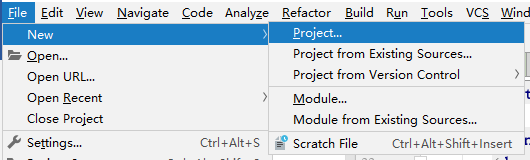
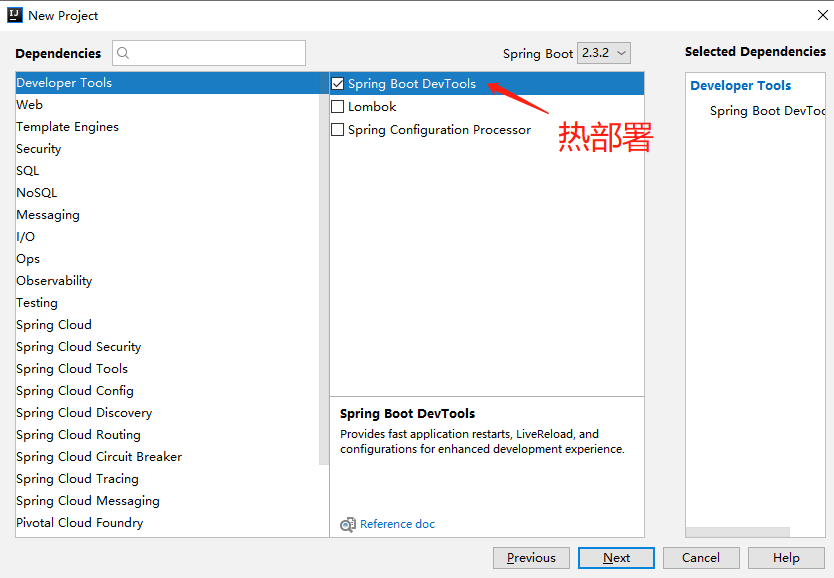
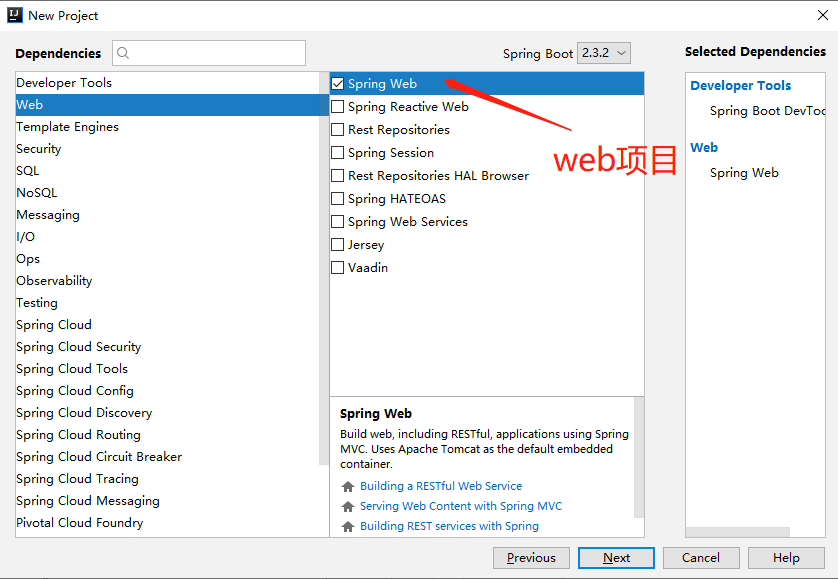
# SpringBoot+Mybatis-Plus快速开始

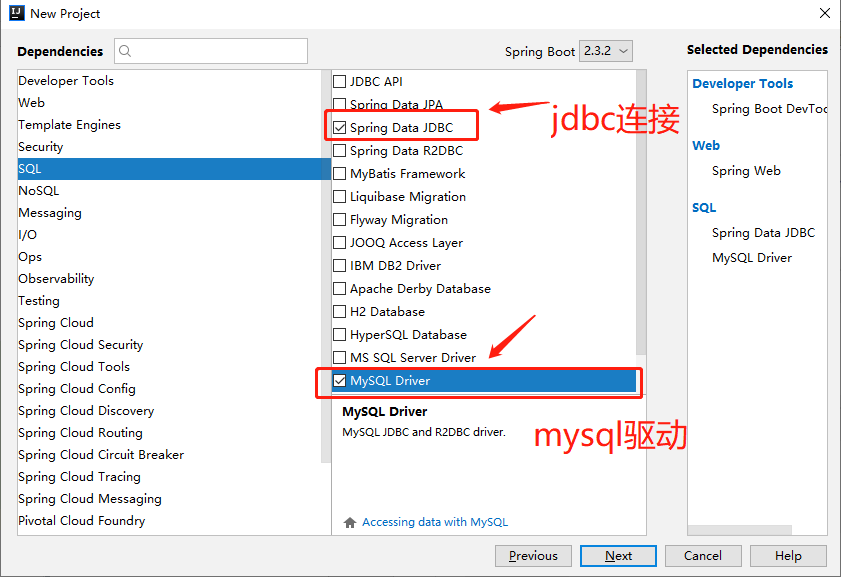
## 一、SpringBoot

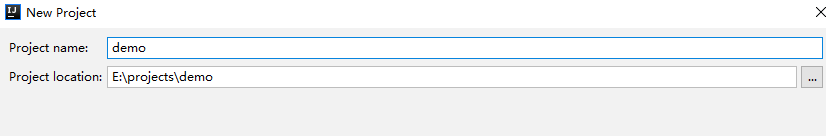
### 1.1 项目创建

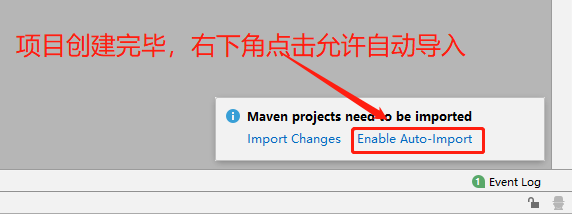
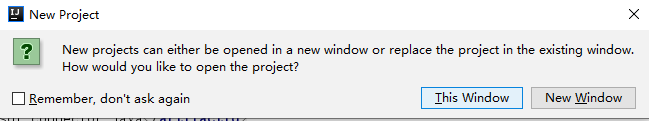




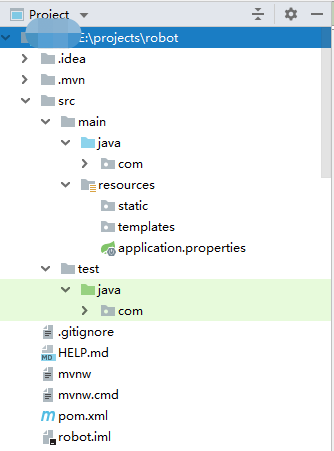








等待创建完毕，目录结构如下：

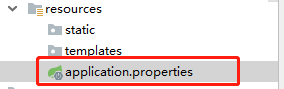


### 1.2 pom.xml文件检查

项目创建完毕后检查pom.xml文件，对应的依赖是否添加正确。

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 <**scope**>runtime</**scope**>  
 <**optional**>true</**optional**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jdbc</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**scope**>runtime</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.junit.vintage</**groupId**>  
 <**artifactId**>junit-vintage-engine</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
</**dependencies**>

### 1.3配置数据库信息



*#数据库信息***spring.datasource.url**=**jdbc:mysql://localhost:3306/数据库名称?serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8&useSSL=true  
spring.datasource.username**=**数据库账号  
spring.datasource.password**=**数据库密码  
spring.datasource.driver-class-name**=**com.mysql.cj.jdbc.Driver**

**备注：上方标红部分需要自行修改，以自己的数据库信息为主。**

### 1.4启动项目

启动项目检查配置是否正确，项目能正常启动即可。



## 二、Mybatis-Plus

### 1.1添加依赖

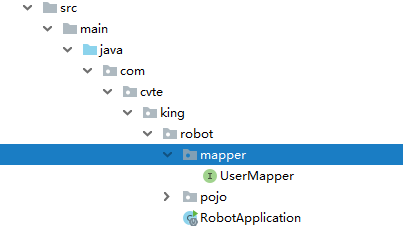
<**dependency**>  
 <**groupId**>com.baomidou</**groupId**>  
 <**artifactId**>mybatis-plus-boot-starter</**artifactId**>  
 <**version**>3.3.2</**version**>  
</**dependency**>

### 1.2配置启动类中的@MapperScan

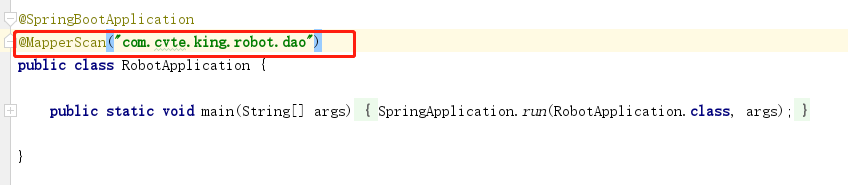
@MapperScan：用于扫描Maaper文件夹，参数指向Mapper文件所在的目录

例子：

我的Mapper文件存放的位置如下：

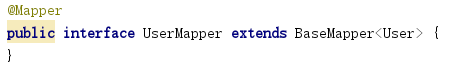


启动类的注解使用方式如下：

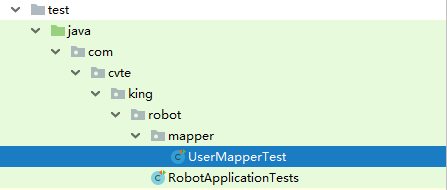


### 1.3编写Mapper类

面向接口编程，接口需要继承BaseMapper<T>接口

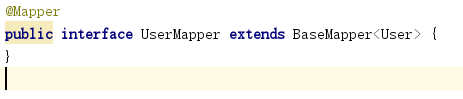


### 1.4编写MapperTest类





### 1.5 Mapper类方法简介



编写的Mapper接口类继承自BaseMapper，BaseMapper默认实现了一些CRUD方法，用于简化对数据库的操作，可查询BaseMapper中的方法来使用，也可以模拟增加自定义方法。

**public interface** BaseMapper<T> **extends** Mapper<T> {  
  
 */\*\*  
 \* 插入一条记录  
 \*  
 \** ***@param entity*** *实体对象  
 \*/* **int** insert(T entity);  
  
 */\*\*  
 \* 根据 ID 删除  
 \*  
 \** ***@param id*** *主键ID  
 \*/* **int** deleteById(Serializable id);  
  
 */\*\*  
 \* 根据 columnMap 条件，删除记录  
 \*  
 \** ***@param columnMap*** *表字段 map 对象  
 \*/* **int** deleteByMap(@Param(Constants.***COLUMN\_MAP***) Map<String, Object> columnMap);  
  
 */\*\*  
 \* 根据 entity 条件，删除记录  
 \*  
 \** ***@param wrapper*** *实体对象封装操作类（可以为 null）  
 \*/* **int** delete(@Param(Constants.***WRAPPER***) Wrapper<T> wrapper);  
  
 */\*\*  
 \* 删除（根据ID 批量删除）  
 \*  
 \** ***@param idList*** *主键ID列表(不能为 null 以及 empty)  
 \*/* **int** deleteBatchIds(@Param(Constants.***COLLECTION***) Collection<? **extends** Serializable> idList);  
  
 */\*\*  
 \* 根据 ID 修改  
 \*  
 \** ***@param entity*** *实体对象  
 \*/* **int** updateById(@Param(Constants.***ENTITY***) T entity);  
  
 */\*\*  
 \* 根据 whereEntity 条件，更新记录  
 \*  
 \** ***@param entity*** *实体对象 (set 条件值,可以为 null)  
 \** ***@param updateWrapper*** *实体对象封装操作类（可以为 null,里面的 entity 用于生成 where 语句）  
 \*/* **int** update(@Param(Constants.***ENTITY***) T entity, @Param(Constants.***WRAPPER***) Wrapper<T> updateWrapper);  
  
 */\*\*  
 \* 根据 ID 查询  
 \*  
 \** ***@param id*** *主键ID  
 \*/* T selectById(Serializable id);  
  
 */\*\*  
 \* 查询（根据ID 批量查询）  
 \*  
 \** ***@param idList*** *主键ID列表(不能为 null 以及 empty)  
 \*/* List<T> selectBatchIds(@Param(Constants.***COLLECTION***) Collection<? **extends** Serializable> idList);  
  
 */\*\*  
 \* 查询（根据 columnMap 条件）  
 \*  
 \** ***@param columnMap*** *表字段 map 对象  
 \*/* List<T> selectByMap(@Param(Constants.***COLUMN\_MAP***) Map<String, Object> columnMap);  
  
 */\*\*  
 \* 根据 entity 条件，查询一条记录  
 \*  
 \** ***@param queryWrapper*** *实体对象封装操作类（可以为 null）  
 \*/* T selectOne(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper);  
  
 */\*\*  
 \* 根据 Wrapper 条件，查询总记录数  
 \*  
 \** ***@param queryWrapper*** *实体对象封装操作类（可以为 null）  
 \*/* Integer selectCount(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper);  
  
 */\*\*  
 \* 根据 entity 条件，查询全部记录  
 \*  
 \** ***@param queryWrapper*** *实体对象封装操作类（可以为 null）  
 \*/* List<T> selectList(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper);  
  
 */\*\*  
 \* 根据 Wrapper 条件，查询全部记录  
 \*  
 \** ***@param queryWrapper*** *实体对象封装操作类（可以为 null）  
 \*/* List<Map<String, Object>> selectMaps(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper);  
  
 */\*\*  
 \* 根据 Wrapper 条件，查询全部记录  
 \* <p>注意： 只返回第一个字段的值</p>  
 \*  
 \** ***@param queryWrapper*** *实体对象封装操作类（可以为 null）  
 \*/* List<Object> selectObjs(@Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper);  
  
 */\*\*  
 \* 根据 entity 条件，查询全部记录（并翻页）  
 \*  
 \** ***@param page*** *分页查询条件（可以为 RowBounds.DEFAULT）  
 \** ***@param queryWrapper*** *实体对象封装操作类（可以为 null）  
 \*/* <E **extends** IPage<T>> E selectPage(E page, @Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper);  
  
 */\*\*  
 \* 根据 Wrapper 条件，查询全部记录（并翻页）  
 \*  
 \** ***@param page*** *分页查询条件  
 \** ***@param queryWrapper*** *实体对象封装操作类  
 \*/* <E **extends** IPage<Map<String, Object>>> E selectMapsPage(E page, @Param(Constants.***WRAPPER***) Wrapper<T> queryWrapper);  
}