尚马教育 JAVA 高级课程

SSM 整合

文档编号：C07

创建日期： 2017-07-07

最后修改日期：2019-10-18

版 本 号：V3.5

电子版文件名：尚马教育-第三阶段-7.ssm 框架整合.docx

文档修改记录：

更新日期	更新作者	更新说明	版本号
2017-07-30	张元林	初始版本	V1.0
2018-07-25	王绍成	Mybatis 版本更新	V2.0
2019-10-18	冯勇涛	课件格式以及课程深度加深	V3.0
2021-02-22	冯勇涛	调整课件	V3.5

目录

1. 整合步骤

添加 spring，springmvc，mybatis 的相关 jar 支持

建立开发目录（handler，service，mapper，entity 等）

配置 springmvc

配置 spring

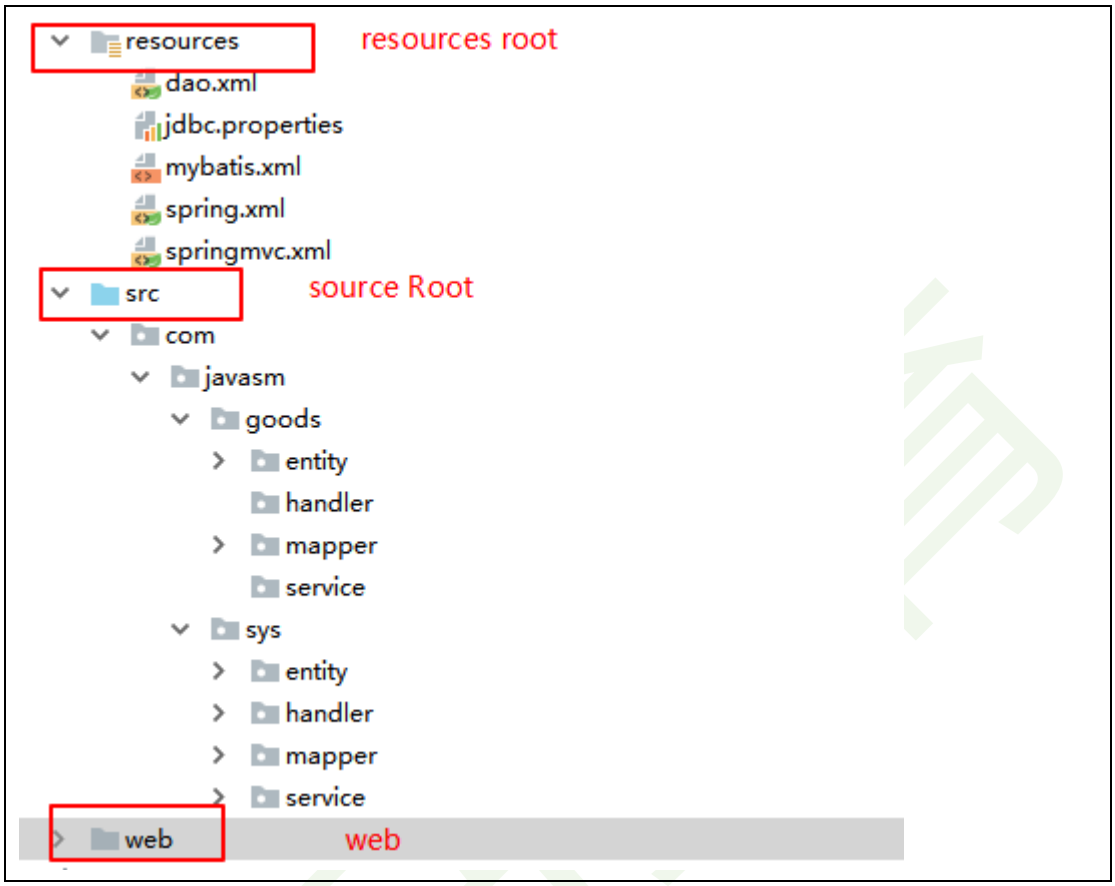
配置 web.xml

配置 mybatis

2. 导入 JAR 包

- 导入 Spring 和 SpringMVC 相关的 Jar
- 导入 JackSon 相关的 Jar(或者 fastjson)
- 导入文件上传的两个 Jar
- 导入 AOP 的 Jar
- 导入 Mybatis 的 Jar 包
- 其他 Jar 包支持
  - 连接池 :druid.jar(获取其他连接池)
  - 日志相关 :commons-logging-1.1.1.jar;
  - mybatis 和 Spring 事务整合 : mybatis-spring-1.3.0.jar;spring-tx;spring-jdbc
  - 数据库链接驱动: mysql-connector-java-5.1.40-bin.jar

3. 建立开发目录



4. 了解两个容器

在 spring-webmvc 包下的 DispatcherServlet 对象中创建的 WebApplicationContext 对象持有了 spring-web 包下的 ContextLoaderListener 对象创建的 WebApplicationContext 对象。

- 把 DispatcherServlet 内创建的容器称为 springmvc 子容器，保存控制层 bean 对象。
- 把 ContextLoaderListener 内创建的容器称为 spring 父容器，保存 service、dao、其他层 bean 对象。

在源码中发现子容器持有父容器，即子容器中的 bean 对象可以依赖父容器中的 bean。换言之，controller 层依赖 service 层。这种设计目的是 springMVC 作为一个控制层框架，职责清晰化，仅管理控制层 bean 对象。

5. 配置 SpringMVC 子容器

在 resources 目录下新建 SpringMVC 配置文件:springmvc.xml

```
<!--spring 注解识别，只识别 Controller 注解，springmvc 控制层框架职责清晰化-->
<context:component-scan base-package="com.javasm" use-default-filters="false">
    <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"></context:include-filter>
</context:component-scan>
<!--springmvc 注解识别-->
<mvc:annotation-driven></mvc:annotation-driven>

<!--文件上传-->
<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="defaultEncoding" value="utf-8"></property>
</bean>
```

6. 配置 Spring 父容器

在 resources 目录下新建 SpringMVC 配置文件:spring.xml

```
<context:component-scan base-package="com.javasm">
    <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"></context:exclude-filter>
</context:component-scan>
```

```
<!--如果我们要向 service 层的对象织入切面，这里做 aop-->
<aop:aspectj-autoproxy></aop:aspectj-autoproxy>
```

7. 配置监听器与前段控制器

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring.xml</param-value>
</context-param>

<!--tomcat 服务器的启动监听器-->
<listener>  <listener-
class>org.springframework.web.context.ContextLoaderListener</lis
tener-class>
</listener>

<!--springMVC 前端控制器-->
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet
-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
```

```
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

8. 配置 MyBatis

8.1. 主配置

创建 jdbc.properties 文件，维护数据库连接信息。

创建 dao.xml 的配置文件,也是一个 spring 的配置文件，专门存放 mybatis 和 spring 的整合配置。在 spring.xml 中通过 import 标签引入 dao.xml 即可。

Jdbc.properties 如下：

```
jdbc.user = root
jdbc.password = root
jdbc.url =
jdbc:mysql://127.0.0.1:3306/704a?characterEncoding=utf8&useSSL=t
rue
jdbc.driver = com.mysql.jdbc.Driver
jdbc.initSize =3
```

Spring.xml 如下：

```
<!--mybatis 集成文件-->
<import resource="classpath:dao.xml"></import>
```

dao.xml 如下：

```
<!--加载 jdbc.properties 文件到 spring 上下文-->
<context:property-placeholder
```

```
location="classpath:jdbc.properties" ignore-
unresolvable="true"></context:property-placeholder>

<!--创建数据库连接池对象-->
<bean id="dataSource"
class="com.alibaba.druid.pool.DruidDataSource" init-
method="init" destroy-method="close">
    <property name="url" value="${jdbc.url}"></property>
    <property name="driverClassName"
value="${jdbc.driver}"></property>
    <property name="username" value="${jdbc.user}"></property>
    <property name="password"
value="${jdbc.password}"></property>
    <property name="initialSize"
value="${jdbc.initSize}"></property>
</bean>

<!--当该 bean 对象被创建出来以后，立即执行 afterPropertiesSet 方法，初始
化 SqlSessionFactory 到 spring 上下文-->
<bean id="sessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
    <property name="mapperLocations"
value="classpath:com/javasm/*/mapper/*.xml"> </property>
    <property name="typeAliasesPackage"
value="com.javasm"></property>
    <property name="configLocation"
value="classpath:mybatis.xml"></property>
</bean>
```

```
<!--对指定包下的接口创建代理对象，注册到 spring 上下文-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage"
value="com.javasm.*.mapper"></property>
</bean>
```

8.2. Druid 连接池完整配置

```
<!-- 数据源 Druid-->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
init-method="init" destroy-method="close">
    <property name="driverClassName" value="${jdbc.driver}"></property>
    <property name="url" value="${jdbc.url}"></property>
    <property name="username" value="${jdbc.username}"></property>
    <property name="password" value="${jdbc.password}"></property>
    <property name="validationQuery" value="true"></property>
    <!-- 配置初始化大小、最小、最大 -->
    <property name="initialSize" value="2" />
    <property name="minIdle" value="1" />
    <property name="maxActive" value="50" />
    <!-- 配置获取连接等待超时的时间 -->
    <property name="maxWait" value="60000" />
    <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
    <property name="timeBetweenEvictionRunsMillis" value="60000" />
    <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
    <property name="minEvictableIdleTimeMillis" value="300000" />
    <property name="testWhileIdle" value="true" />
    <property name="testOnBorrow" value="false" />
    <property name="testOnReturn" value="false" />
</bean>
```

8.3. 配置事务管理器

```
<!--配置事务管理器对象，id 必须是 transactionManager-->
<bean                                                id="transactionManager"
```

```
class="org.springframework.jdbc.datasource.DataSourceTransactionManage
r">
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

8.3.1. 注解式事务

开启事务注解 *Transactional* 识别，该注解使用在 service 层的方法上。

```
<!--依赖于 id 为 transactionManager 的事务管理器对象，对带有
@Transactional 注解的方法做织入-->
<tx:annotation-driven></tx:annotation-driven>
```

8.3.2. 配置式事务

```
<!--事务通知，以及事务生效规则-->
<tx:advice id="txAdvice">
    <tx:attributes>
        <tx:method name="add*" propagation="REQUIRED"/>
        <tx:method name="insert*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="del*" propagation="REQUIRED"/>
        <tx:method                                    name="select*"
propagation="SUPPORTS"></tx:method>
        <tx:method                                     name="query*"
propagation="SUPPORTS"></tx:method>
    </tx:attributes>
</tx:advice>
<!--事务通知织入到切入点中-->
<aop:config>
    <aop:pointcut  id="servicePointcut"  expression="execution(*
```

```
com.javasm.*.service.*.*(..))"></aop:pointcut>
    <aop:advisor          advice-ref="txAdvice"          pointcut-
ref="servicePointcut"></aop:advisor>
</aop:config>
```

9. 配置跨域过滤器

在 resources 下创建 cors.xml，并导入到 spring.xml 中。

9.1. 注册 CorsFilter 到 spring 上下文

```
<bean id="corsFilter"
class="org.springframework.web.filter.CorsFilter">
    <constructor-arg name="configSource">
    <bean
class="org.springframework.web.cors.UrlBasedCorsConfigurationSou
rce">
        <property name="corsConfigurations">
            <map>
                <entry key="/**">
                    <bean
class="org.springframework.web.cors.CorsConfiguration">
                        <property name="allowCredentials"
value="true"/>
                        <property name="allowedMethods">
                            <list>
                                <value>GET</value>
                                <value>POST</value>
                                <value>HEAD</value>
                                <value>PUT</value>
```

```
                        <value>DELETE</value>
                        <value>OPTIONS</value>
                    </list>
                </property>
                <property name="allowedHeaders" value="*"/>
                <property name="allowedOrigins" value="*"/>
            </bean>
        </entry>
    </map>
</property>
</bean>
</constructor-arg>
</bean>
```

9.2. 配置 web 过滤器

在 web.xml 中配置 CorsFilter

```
<filter>
    <filter-name>myCorsFilter</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filt
er-class>
    <init-param>
        <param-name>targetBeanName</param-name>
        <param-value>corsFilter</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>myCorsFilter</filter-name>
```

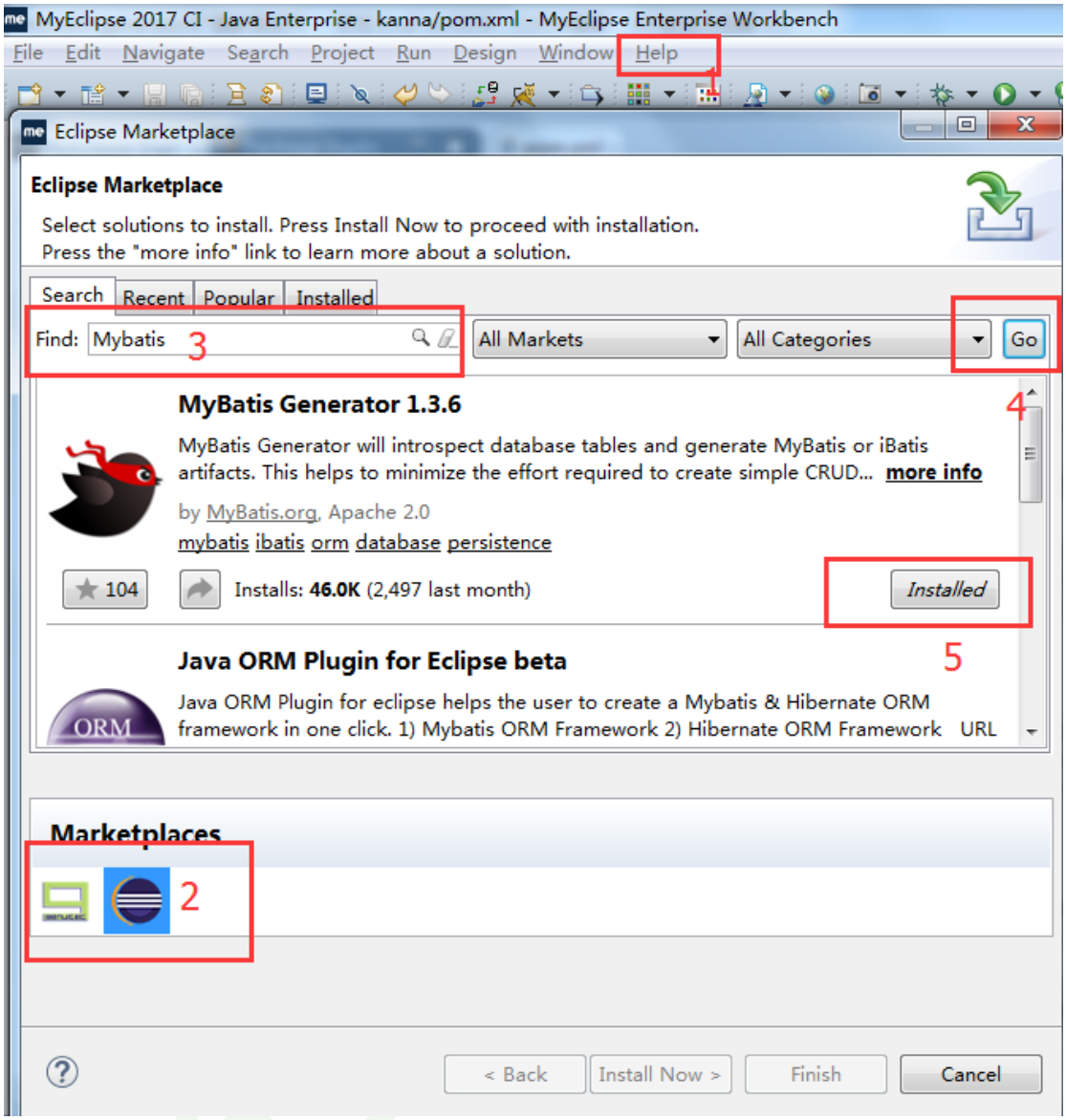
```
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

10. Mybatis Generator 配置

10. 1. MyEclipse

10. 1. 1. 安装

打开 myeclipse--help---Install from catalog--选择 eclipse 应用市场---Find 中填入 Mybatis (注意: 区分大小写) 点 go 搜索出 Mybatis Generator 选择安装. 关闭 myeclipse 重新打开.



10.1.2. 使用

创建 generatorConfig.xml 文件

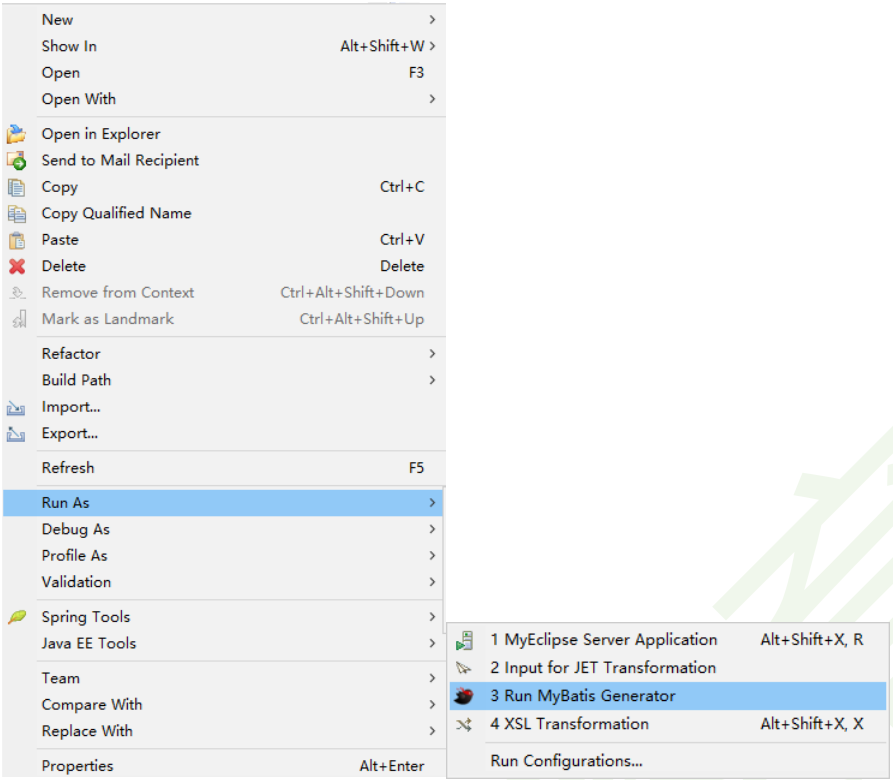
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD
MyBatis Generator Configuration 1.0//EN"
"http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
    <!-- 数据库驱动包位置 -->
    <classPathEntry location="D:\generator\mysql-connector-java-
```

```
5.1.34.jar" />
   <context id="DB2Tables" targetRuntime="MyBatis3">
      <commentGenerator>
         <property name="suppressAllComments" value="true" />
      </commentGenerator>
      <!-- 数据库链接 URL、用户名、密码 -->
      <jdbcConnection driverClass="com.mysql.jdbc.Driver"
connectionURL="jdbc:mysql://localhost:3306/704a" userId="root"
password="root">
      </jdbcConnection>

      <javaTypeResolver>
         <property name="forceBigDecimals" value="false" />
      </javaTypeResolver>

      <!-- 生成模型的包名和位置 -->
      <javaModelGenerator targetPackage="com.javasm.goods.entity"
targetProject="D:\threeWorkspace\1111ssm\src">
         <property name="enableSubPackages" value="true" />
         <property name="trimStrings" value="true" />
      </javaModelGenerator>

      <!-- 生成的映射文件包名和位置 -->
      <sqlMapGenerator targetPackage="com.javasm.goods.mapper"
targetProject="D:\threeWorkspace\1111ssm\src">
         <property name="enableSubPackages" value="true" />
      </sqlMapGenerator>

      <!-- 生成 DAO 的包名和位置 -->
```

```
      <javaClientGenerator type="XMLMAPPER"
targetPackage="com.javasm.goods.mapper"
targetProject="D:\threeWorkspace\1111ssm\src">
        <property name="enableSubPackages" value="true" />
      </javaClientGenerator>

      <!-- 要生成那些表(更改 tableName 和 domainObjectName 就可以) -->
      <table tableName="goods" domainObjectName="Goods"
enableCountByExample="false" enableUpdateByExample="false"
enableDeleteByExample="false" enableSelectByExample="false"
selectByExampleQueryId="false" />
      <table tableName="goodstype" domainObjectName="GoodsType"
enableCountByExample="false" enableUpdateByExample="false"
enableDeleteByExample="false" enableSelectByExample="false"
selectByExampleQueryId="false" />
   </context>
</generatorConfiguration>
```

10.1.3. 运行

右键点击配置文件，选择运行，



10.2. IDEA

10.2.1. 安装

因为 IntelliJ 中没有 mybatis-generato 对应的插件，所以需要在 MAVEN 中使用 mybatis-generator-maven-plugin 插件来完成功能或者运行 mybatis Generator 代码方式来生成。

代码方式：

第一步：创建独立的工程：mybatisGenerator

第二步：工程添加依赖项：mybatis-generator-core.jar 与 mysql-connector-java.jar

第三步：拷贝 **generator.xml 配置文件到 src 目录**

第三步：从官网复制生成代码如下：

```
public class MyGenerator {

    public static void main(String[] args) throws Exception {

        List<String> warnings = new ArrayList<String>();

        boolean overwrite = true;

        String path =
```

```
MyGenerator.class.getClassLoader().getResource("generator.xml").
getPath();
        File configFile = new File(path);
        ConfigurationParser cp = new
ConfigurationParser(warnings);
        Configuration config = cp.parseConfiguration(configFile);
        DefaultShellCallback callback = new
DefaultShellCallback(overwrite);
        MyBatisGenerator myBatisGenerator = new
MyBatisGenerator(config, callback, warnings);
        myBatisGenerator.generate(null);
    }
}
```

11. PageHelper 分页插件

只需要引入两个 jar

jsqlparser-0.9.5.jar

pagehelper-5.0.0.jar

11.1. 配置 dao.xml 文件

```
<bean id="sessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
    <property name="mapperLocations"
value="classpath:com/javasm/*/mapper/*.xml"></property>
    <property name="typeAliasesPackage"
```

```
value="com.javasm"></property>
    <property name="configLocation"
value="classpath:mybatis.xml"></property>
    <property name="plugins">
        <array>
            <bean class="com.github.pagehelper.PageInterceptor">
                <!--分页参数合理化-->
                <property name="properties">
                    <value>reasonable=true</value>
                </property>
            </bean>
        </array>
    </property>
</bean>
```

在原来配置文件上加入红色代码

11.2. Mapper.xml

```
<select id="selectList" resultMap="BaseResultMap">
    select * from t_student
```

只需要写基本的查询，不需要加入分页信息

11.3. Handler

```
@GetMapping("users")
public ResponseEntity selectUsers(Sysuser user,
@RequestParam(defaultValue = "1") Integer pageNum,
@RequestParam(defaultValue = "3") Integer pageSize){
    //在调用服务层的集合查询之前，进行分页开启
```

```
    PageHelper.startPage(pageNum,pageSize);
    //ArrayList 类型-->分页后-->Page 类型
    List<Sysuser> sysusers = us.selectUsers(user);
    //封装分页数据以及数据集
    PageInfo info = new PageInfo(sysusers);
    return ResponseEntity.ok(info);
}
```

11.4. 属性介绍

属性名称	属性描述
pageNum	当前页
pageSize	每页的数量
size	当前页的数量
startRow	当前页面第一个元素在数据库中的行号(不常用)
endRow	当前页面最后一个元素在数据库中的行号(不常用)
total	总记录数
pages	总页数
list	结果集
firstPage	第一页
prePage	前一页
isFirstPage	是否为第一页
isLastPage	是否为最后一页
hasPreviousPage	是否有前一页

hasNextPage	是否有下一页
navigatePages	导航页码数
navigatepageNums	导航页码数组

任务：

使用第二阶段项目前端 vue 视图，实现单表的操作。