SSM整合

1、Spring的IOC容器在什么时候创建?

整合就是要使Spring和SpringMVC管理各自的组件，然后和MyBatis整合。

从组件的依赖关系中可知：SpringMVC管理的组件是控制层组件，其他组件（如业务层组件service）由Spring管理，SpringMVC的控制层（controller）组件要依赖于Spring的业务层（service）组件，因为在控制层中要创建一个service的成员变量，来进行一个自动装配，这样就可以在控制层中使用service成员对象了，因此Spring创建IOC的容器要早于SpringMVC创建IOC容器。

SpringMVC的IOC容器实在DispatcherServlet初始化过程中创建的，因此Spring的IOC要在DispatcherServlet初始化之前创建。

服务器中三大组件的执行顺序为：监听器、过滤器、Servlet，因此可以把Spring的IOC创建放在监听器或过滤器中，但是过滤器主要功能是过滤当前的请求和功能，因此我们不能为了实现一个功能就放弃了其原始的功能和意义，因此将Spring的IOC创建放在监听器中。

Spring提供了监听器ContextLoaderListener，实现ServletContextListener接口，可监听ServletContext的状态，在web服务器的启动，读取Spring的配置文件，创建Spring的IOC容器。web应用中必须在web.xml中配置。

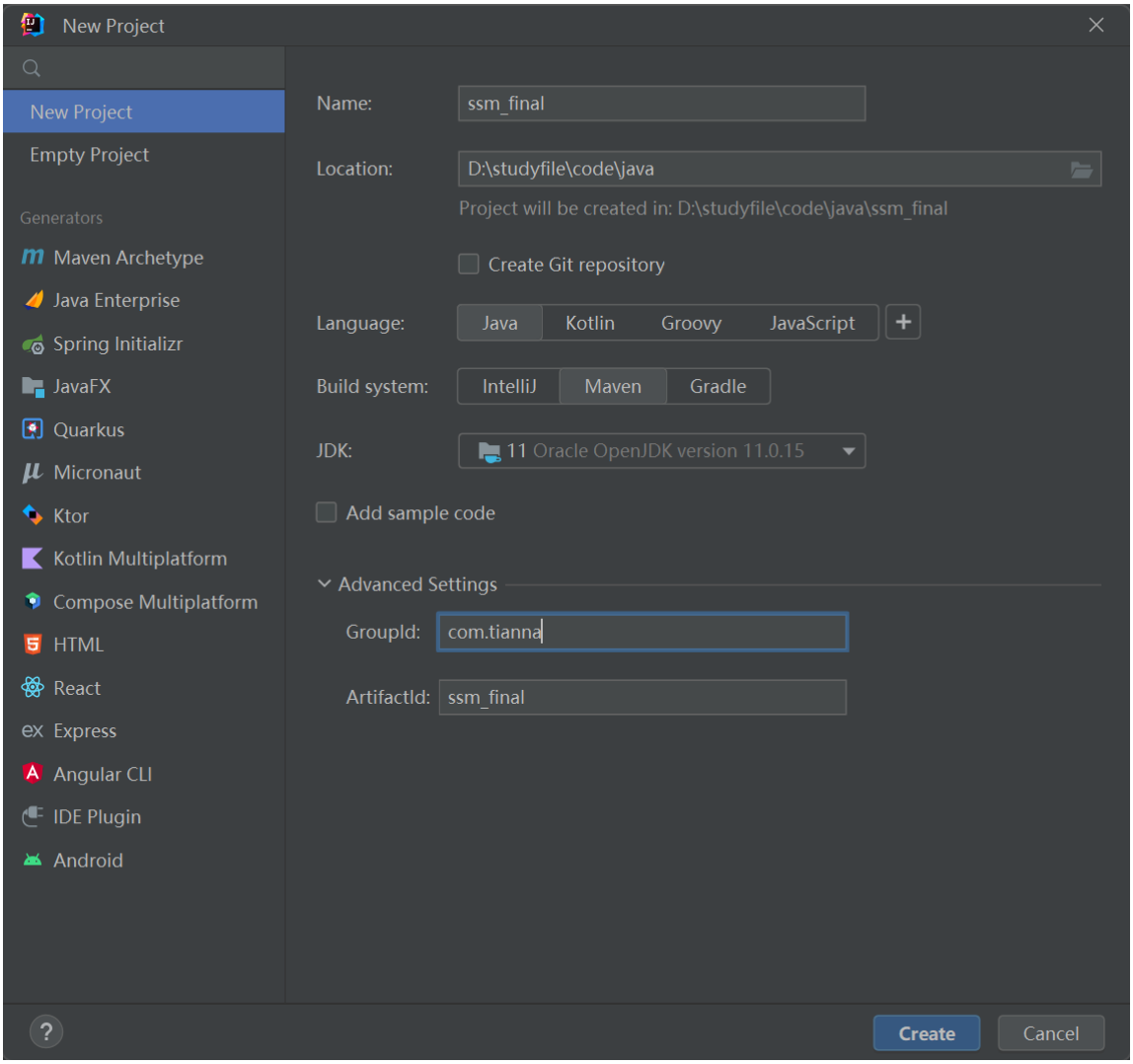
Spring配置文件默认位置和名称：/WEB-INF/applicationContext.xml。我们可通过上下文参数自定义Spring配置文件的位置和名称，在web.xml中详细的配置代码如下：

```
<listener>
    <!--
        在服务器启动时加载Spring的配置文件
        Spring配置文件默认位置和名称：/WEB-INF/applicationContext.xml
        可通过上下文参数自定义Spring配置文件的位置和名称
    -->
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
<!--自定义Spring配置文件的位置和名称-->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring.xml</param-value>
</context-param>
```

Spring的IOC容器为父容器，SpringMVC的IOC容器为子容器，子容器可以访问父容器的bean，父容器无法访问子容器的bean。

2、准备工作

1、创建Maven项目



2、导入依赖

在pom.xml文件中导入如下依赖

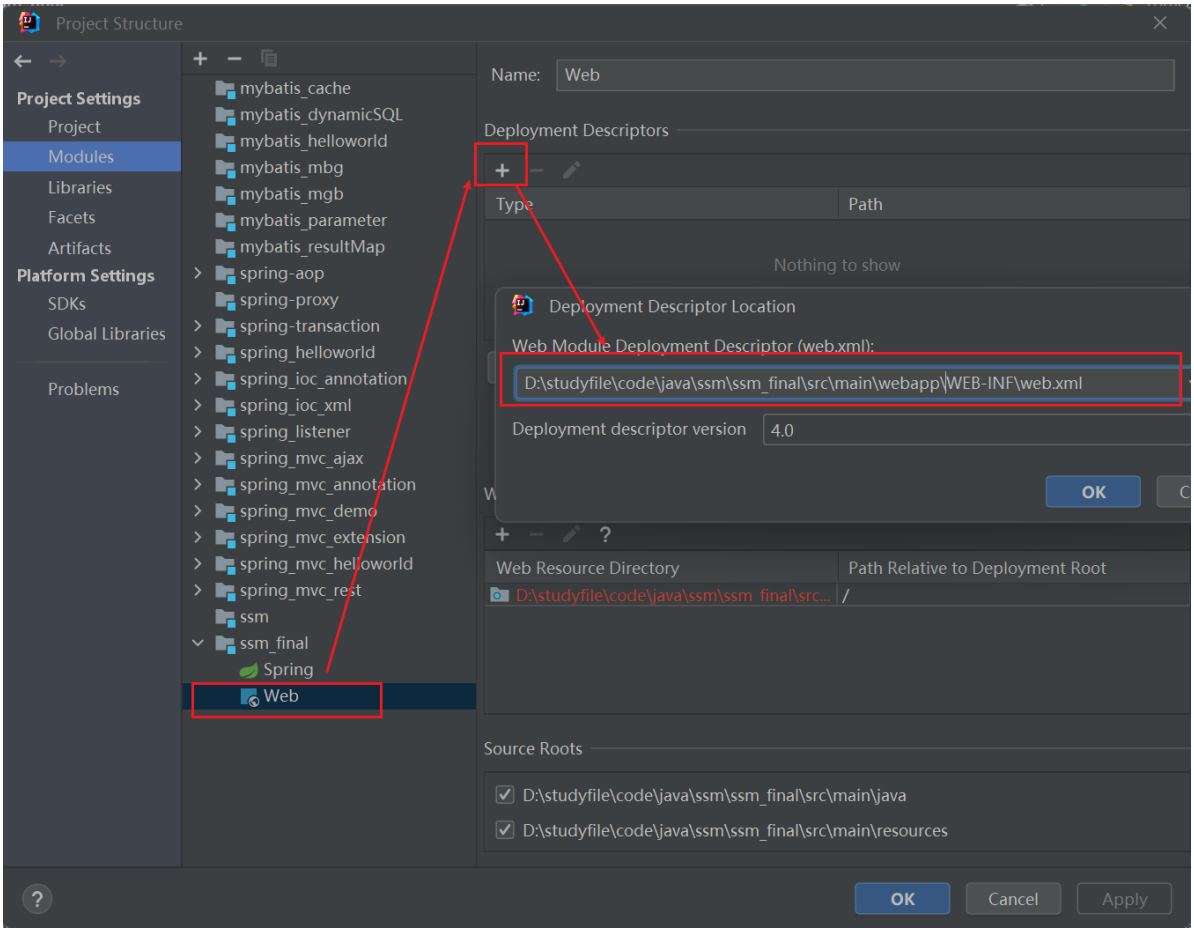
```
<packaging>war</packaging>

<properties>
    <!--自定义属性，spring版本-->
    <spring.version>5.3.1</spring.version>
</properties>

<dependencies>
    <!--spring-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!--springmvc-->
```

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!--主要使用事务管理器-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!--管理切面-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aspects</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!-- Mybatis核心 -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.7</version>
    </dependency>
    <!--mybatis和spring的整合包-->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>2.0.6</version>
    </dependency>
    <!-- 连接池 -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.0.9</version>
    </dependency>
    <!-- junit测试 -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
```

```
    </dependency>
    <!-- MySQL驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.16</version>
    </dependency>
    <!-- log4j日志 -->
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
    <!--
https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->
    <!--分页插件-->
    <dependency>
        <groupId>com.github.pagehelper</groupId>
        <artifactId>pagehelper</artifactId>
        <version>5.2.0</version>
    </dependency>
    <!-- 日志 -->
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.2.3</version>
    </dependency>
    <!-- ServletAPI -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
    <!--处理json的包-->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.12.1</version>
    </dependency>
    <!--文件上传依赖-->
    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.3.1</version>
    </dependency>
    <!-- Spring5和Thymeleaf整合包 -->
    <dependency>
        <groupId>org.thymeleaf</groupId>
        <artifactId>thymeleaf-spring5</artifactId>
        <version>3.0.12.RELEASE</version>
```

```
    </dependency>
</dependencies>
```

3、为当前工程添加web模块



4、创建表

创建一个员工表，sql代码如下：

```
CREATE TABLE `t_emp1` (
`emp_id` int(11) NOT NULL AUTO_INCREMENT,
`emp_name` varchar(20) DEFAULT NULL,
`age` int(11) DEFAULT NULL,
`sex` char(1) DEFAULT NULL,
`email` varchar(50) DEFAULT NULL,
PRIMARY KEY (`emp_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

3、配置web.xml

web.xml的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
```

```
    <!--配置Spring的编码过滤器，要放在过滤器的第一个-->
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <!--配置处理请求方式的过滤器-->
    <filter>
        <filter-name>HiddenHttpMethodFilter</filter-name>
        <filter-
class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>HiddenHttpMethodFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <!--配置SpringMVC的前端控制器DispatcherServlet-->
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!--设置SpringMVC配置文件自定义的位置和名称-->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <!--将DispatcherServlet的初始化提前到服务器启动时-->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!--配置Spring的监听器，在服务器启动时加载Spring的配置文件-->
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
```

```
    </listener>
    <!--设置Spring配置文件自定义的位置和名称-->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring.xml</param-value>
    </context-param>
</web-app>
```

4、创建SpringMVC的配置文件并配置

在resources目录下创建名字为springmvc的配置文件，并对其进行配置，配置内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!--扫描控制层组件-->
    <context:component-scan base-package="com.tianna.ssm.controller">
</context:component-scan>

    <!--配置视图解析器-->
    <bean id="viewResolver"
          class="org.thymeleaf.spring5.view.ThymeleafViewResolver">
        <property name="order" value="1"/>
        <property name="characterEncoding" value="UTF-8"/>
        <property name="templateEngine">
            <bean class="org.thymeleaf.spring5.SpringTemplateEngine">
                <property name="templateResolver">
                    <bean

 class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolv
er">
                        <!-- 视图前缀 -->
                        <property name="prefix" value="/WEB-
INF/templates/"/>
                        <!-- 视图后缀 -->
                        <property name="suffix" value=".html"/>
                        <property name="templateMode" value="HTML5"/>
                        <property name="characterEncoding" value="UTF-8" />
                    </bean>
                </property>
            </bean>
        </property>
    </bean>
```

```
    <!--配置默认的servlet处理静态资源-->
    <mvc:default-servlet-handler/>

    <!--开启mvc的注解驱动（要加）-->
    <mvc:annotation-driven/>

    <!--配置视图控制器-->
    <!--index.html的视图解析器-->
    <mvc:view-controller path="/" view-name="index"></mvc:view-controller>

    <!--配置文件上传解析器 id必须为multipartResolver-->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/
>

    <!--拦截器和异常解析器根据自己需要添加-->
</beans>
```

controller层组件由SpringMVC管理，在java目录下创建一个名为`com.tianna.ssm.controller`的包，并在该包下创建一个名为`EmployeeController`的控制器。

```
@Controller
public class EmployeeController {
}
```

在webapp/WEB-INF下创建templates文件夹，用于存放视图文件，并在该文件夹下创建一个名为index.html的首页：

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
<h1>index.html</h1>
</body>
</html>
```

5、配置MyBatis相关的内容

1、创建属性文件jdbc.properties

在resources目录下创建属性文件jdbc.properties，内容如下：

```
jdbc.driver = com.mysql.cj.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/ssm?serverTimezone=UTC
jdbc.username = root
jdbc.password = 123456
```

2、创建MyBatis的核心配置文件mybatis-config.xml

在resources目录下创建MyBatis核心配置文件mybatis-config.xml，内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!--开启驼峰命名自动映射-->
    <settings>
        <setting name="mapUnderscoreToCamelCase" value="true"/>
    </settings>

    <!--设置类型别名所配置的包-->
    <typeAliases>
        <package name="com.tianna.ssm.pojo"/>
    </typeAliases>

    <plugins>
        <!--配置分页插件-->
        <plugin interceptor="com.github.pagehelper.PageInterceptor">
</plugin>
    </plugins>
</configuration>
```

3、创建Mapper接口和映射文件

创建接口：

在com.tianna.ssm包下创建一个名为mapper的包，并在该包下创建一个名为EmployeeMapper的Mapper接口，内容如下：

```
package com.tianna.ssm.mapper;

/**
 * @author tiancn
 * @date 2022/8/22 22:57
 */
public interface EmployeeMapper {
}
```

创建mapper映射文件：

在resources目录下创建一个与mapper接口相同路径的文件夹（多层文件夹），名为`com/tianna/ssm/mapper`，在该文件夹下创建一个名为EmployeeMapper.xml（与mapper接口的名字一样）的映射文件，内容如下：

```
<?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.tianna.ssm.mapper.EmployeeMapper">

</mapper>
```

4、创建日志文件log4j.xml

在resources目录下创建一个名为log4j.xml的日志i文件，内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
    <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
        <param name="Encoding" value="UTF-8" />
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-5p %d{MM-dd
HH:mm:ss,SSS} %m (%F:%L) \n" />
        </layout>
    </appender>
    <logger name="java.sql">
        <level value="debug" />
    </logger>
    <logger name="org.apache.ibatis">
        <level value="info" />
    </logger>
    <root>
        <level value="debug" />
        <appender-ref ref="STDOUT" />
    </root>
</log4j:configuration>
```

6、创建Spring的配置文件并配置

在resources目录下创建一个名为spring.xml的Spring配置文件，其内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
```

```
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd">
    <!--扫描组件（除控制层）-->
    <context:component-scan base-package="com.tianna.ssm">
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

    <!--引入jdbc.properties-->
    <context:property-placeholder location="classpath:jdbc.properties">
</context:property-placeholder>

    <!--配置数据源-->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <!--配置事务管理器-->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <!--
        开启事务的注解驱动
        将使用注解@Transactional标识的方法或类中所有的方法进行事务管理
    -->
    <tx:annotation-driven transaction-manager="transactionManager"/>

    <!--配置SqlSessionFactoryBean，可以直接在Spring的IOC容器中获取
SqlSessionFactory-->
    <bean class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--设置Mybatis的核心配置文件-->
        <property name="configLocation" value="classpath:mybatis-
config.xml"></property>
        <!--设置数据源 其他内容都可以在mybatis-config.xml中配置-->
        <property name="dataSource" ref="dataSource"></property>
        <!--设置映射文件的路径，只有映射文件的包和mapper接口的包不一致时需要设
置-->
        <!--<property name="mapperLocations"
value="classpath:mappers/*.xml"></property>-->
    </bean>

    <!--
```

```
        配置mapper接口的扫描，可以将包下所有的mapper接口，通过SqlSession创建代理实现类对象，并将这些对象交给IOC容器管理。
    -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.tianna.ssm.mapper">
</property>
    </bean>

</beans>
```

在java目录下的`com.tianna.ssm`包下创建一个`service`的包，在该包下创建一个名为EmployeeService的接口，内容如下：

```
package com.tianna.ssm.service;

/**
 * @author tiancn
 * @date 2022/8/22 22:47
 */
public interface EmployeeService {
}
```

在java目录下的`com.tianna.ssm.service`包下`impl`的包，在该包下创建一个EmployeeService接口的实现类`EmployeeServiceImpl`，内容如下：

```
package com.tianna.ssm.service.impl;

import com.tianna.ssm.service.EmployeeService;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/**
 * @author tiancn
 * @date 2022/8/22 22:47
 */
@Service
@Transactional
public class EmployeeServiceImpl implements EmployeeService {

}
```

在java目录下的`com.tianna.ssm`包下创建一个`pojo`的包，并在该包下创建一个实体类`Employee`，内容如下：

```
package com.tianna.ssm.pojo;

/**
 * @author tiancn
 * @date 2022/8/22 23:16
```

```
 */

public class Employee {
    private Integer empId;
    private String empName;
    private Integer age;
    private String sex;
    private String email;

    public Employee() {
    }

    public Employee(Integer empId, String empName, Integer age, String sex,
String email) {
        this.empId = empId;
        this.empName = empName;
        this.age = age;
        this.sex = sex;
        this.email = email;
    }

    public Integer getEmpId() {
        return empId;
    }

    public void setEmpId(Integer empId) {
        this.empId = empId;
    }

    public String getEmpName() {
        return empName;
    }

    public void setEmpName(String empName) {
        this.empName = empName;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
```

```
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "empId=" + empId +
                ", empName='" + empName + '\'' +
                ", age=" + age +
                ", sex='" + sex + '\'' +
                ", email='" + email + '\'' +
                '}';
    }
}
```

7、测试功能

下面使用SSM实现员工信息的管理，主要包括员工信息分页查询，员工信息添加，员工信息修改，员工信息删除。

7.1、控制层

控制层`EmployeeControlle`r代码如下：

```
/**
 * @author tiancn
 * @date 2022/8/22 22:24
 * 查询所有的员工信息-->/employee-->get
 * 查询员工的分页信息-->/employee/page/1-->get
 * 根据id查询员工信息-->/employee/1-->get
 * 跳转到添加页面-->/to/add-->get
 * 添加员工信息-->/employee-->post
 * 修改员工信息-->/employee-->put
 * 删除员工信息-->/employee-->delete
 */
@Controller
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    //获取员工的分页信息
    @RequestMapping(value = "/employee/page/{pageNum}",method =
RequestMethod.GET)
```

```
    public String getEmployeePage(@PathVariable("pageNum") Integer pageNum,
Model model){
        //获取员工的分页信息
        PageInfo<Employee> page = employeeService.getEmployeePage(pageNum);
        //将分页数据共享到请求域中
        model.addAttribute("page",page);
        //跳转到employee_list
        return "employee_list";

    }
    //跳转到添加员工界面
    @RequestMapping(value = "/to/add",method = RequestMethod.GET)
    public String toAddEmployeeView(){
        return "employee_add";
    }
    //添加员工
    @RequestMapping(value = "/employee",method = RequestMethod.POST)
    public String addEmployee(Employee employee){
        employeeService.insertEmployee(employee);
        //暂时先跳转到第一页把
        return "redirect:/employee/page/1";

    }
    @RequestMapping(value = "/employee/{id}",method = RequestMethod.GET)
    public String toUpdate(@PathVariable("id") Integer id,Model model){
        //根据id查询员工信息
        Employee employee = employeeService.getEmployeeById(id);
        //将员工信息共享到请求域中
        model.addAttribute("employee",employee);
        //跳转到employee_update
        return "employee_update";

    }
    @RequestMapping(value = "/employee",method = RequestMethod.PUT)
    public String updateEmployee(Employee employee){
        System.out.println(employee);
        //更新员工信息
        employeeService.updateEmployee(employee);
        //暂时先跳转到第一页把
        return "redirect:/employee/page/1";
    }
    @RequestMapping(value = "/employee/{id}" ,method =
RequestMethod.DELETE)
    public String deleteEmployee(@PathVariable("id") Integer id){
        employeeService.deleteEmployee(id);
        //暂时先跳转到第一页把
        return "redirect:/employee/page/1";

    }
}
```

7.2、服务层

服务层接口EmployeeService代码如下：

```
/**
 * @author tiancn
 * @date 2022/8/22 22:47
 */
public interface EmployeeService {
    /**
     * 查询所有的员工信息
     * @return
     */
    List<Employee> getAllEmployee();

    /**
     * 获取员工的分页信息
     * @param pageNum
     * @return
     */
    PageInfo<Employee> getEmployeePage(Integer pageNum);

    /**
     * 添加员工信息
     * @param employee
     */
    void insertEmployee(Employee employee);

    /**
     * 根据id查询员工信息
     * @param id
     * @return
     */
    Employee getEmployeeById(Integer id);

    /**
     * 更新员工信息
     * @param employee
     */
    void updateEmployee(Employee employee);

    /**
     * 删除员工信息
     * @param id
     */
    void deleteEmployee(Integer id);
}
```

服务层接口实现类EmployeeServiceImpl代码如下：

```
/**
```

```
 * @author tiancn
 * @date 2022/8/22 22:47
 */
@Service
@Transactional
public class EmployeeServiceImpl implements EmployeeService {
    @Autowired
    private EmployeeMapper employeeMapper;

    @Override
    public List<Employee> getAllEmployee() {
        return employeeMapper.getAllEmployee();
    }

    @Override
    public PageInfo<Employee> getEmployeePage(Integer pageNum) {
        //开启分页功能
        PageHelper.startPage(pageNum,3);
        //查询所有的员工信息
        List<Employee> list = employeeMapper.getAllEmployee();
        //获取分页相关数据
        PageInfo<Employee> page = new PageInfo<>(list,3);
        return page;
    }

    @Override
    public void insertEmployee(Employee employee) {
        employeeMapper.insertEmployee(employee);
    }

    @Override
    public Employee getEmployeeById(Integer id) {
        return employeeMapper.getEmployeeById(id);
    }

    @Override
    public void updateEmployee(Employee employee) {
        employeeMapper.updateEmployee(employee);
    }

    @Override
    public void deleteEmployee(Integer id) {
        employeeMapper.deleteEmployee(id);
    }
}
```

7.3、Mapper接口和映射文件

mapper接口EmployeeMapper代码如下：

```
/**
 * @author tiancn
 * @date 2022/8/22 22:57
 */
public interface EmployeeMapper {
    /**
     * 查询所有的员工信息
     * @return
     */
    List<Employee> getAllEmployee();

    /**
     * 添加员工信息
     * @param employee
     */
    void insertEmployee(Employee employee);

    /**
     * 根据id查询员工信息
     * @param id
     * @return
     */
    Employee getEmployeeById(@Param("id") Integer id);

    /**
     * 修改员工信息
     * @param employee
     */
    void updateEmployee(Employee employee);

    /**
     * 删除员工信息
     * @param id
     */
    void deleteEmployee(@Param("id") Integer id);
}
```

mapper接口映射文件代码如下：

```
<?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.tianna.ssm.mapper.EmployeeMapper">
    <!--查询所有员工信息-->
    <!--List<Employee> getAllEmployee();-->
    <select id="getAllEmployee" resultType="employee">
```

```
        select * from t_emp1
    </select>
    <!--添加员工信息-->
    <!--void insertEmployee(Employee employee);-->
    <insert id="insertEmployee">
        insert into t_emp1 values(null,#{empName},#{age},#{sex},#{email})
    </insert>
    <!--根据id查询用户信息-->
    <!--Employee getEmployeeById(@Param("id") Integer id);-->
    <select id="getEmployeeById" resultType="employee">
        select * from t_emp1 where emp_id = #{id}
    </select>
    <!--修改员工信息-->
    <!--void updateEmployee(Employee employee);-->
    <update id="updateEmployee">
        update t_emp1 set emp_name = #{empName},age = #{age},sex = #
{sex},email = #{email} where emp_id = #{empId}
    </update>
    <!--删除员工信息-->
    <!--void deleteEmployee(@Param("id") Integer id);-->
    <delete id="deleteEmployee">
        delete from t_emp1 where emp_id = #{id}
    </delete>
</mapper>
```

7.4、前端页面

1、首页index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
<h1>index.html</h1>
<a th:href="@{/employee/page/1}">查询员工的分页信息</a>
</body>
</html>
```

2、员工信息展示页面employee_list.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>员工列表</title>
```

```
    <link rel="stylesheet" th:href="@{/static/css/index_work.css}">
</head>
<body>
<div id="app">
    <table>
        <tr>
            <th colspan="6">员工列表</th>
        </tr>
        <tr>
            <th>流水号</th>
            <th>员工姓名</th>
            <th>年龄</th>
            <th>性别</th>
            <th>邮箱</th>
            <th>操作（<a th:href="@{/to/add}">添加</a>）</th>
        </tr>
        <tr th:each = "employee,status:${page.list}">
            <td th:text="${status.count}"></td>
            <td th:text="${employee.empName}"></td>
            <td th:text="${employee.age}"></td>
            <td th:text="${employee.sex}"></td>
            <td th:text="${employee.email}"></td>
            <td>
                <a @click = "deleteEmployee" th:href="@{'/employee/' +
${employee.empId}}">删除</a>
                <a th:href="@{'/employee/' + ${employee.empId}}">修改</a>
            </td>
        </tr>
    </table>
    <div style="text-align: center;">
        <a th:if="${page.hasPreviousPage}" th:href="@{/employee/page/1}">首
页</a>
        <a th:if="${page.hasPreviousPage}" th:href="@{'/employee/page/' +
${page.prePage}}">上一页</a>
        <span th:each="num : ${page.navigatepageNums}">
        <a th:if="${page.pageNum == num}" style="color: red"
th:href="@{'/employee/page/' + ${num}}" th:text="'[' + ${num} + ']'"></a>
        <a th:if="${page.pageNum != num}" th:href="@{'/employee/page/' +
${num}}" th:text="${num}"></a>
    </span>
        <a th:if="${page.hasNextPage}"
th:href="@{'/employee/page/'+${page.nextPage}}">下一页</a>
        <a th:if="${page.hasNextPage}" th:href="@{'/employee/page/' +
${page.pages}}">末页</a>
    </div>
    <form method="post">
        <input type="hidden" name="_method" value = "delete">
    </form>
</div>

<script type="text/javascript" th:src="@{/static/js/vue.js}"></script>
```

```
<script type="text/javascript">
    var vue = new Vue({
        el:"#app",
        methods:{
            deleteEmployee(){
                //获取form表单
                var form = document.getElementsByTagName("form")[0];
                //将超链接的href属性值赋值给form表单的action属性
                //event.target表示当前触发事件的标签
                form.action = event.target.href;
                //表单提交
                form.submit();
                //组织超链接的默认行为
                event.preventDefault();
            }
        }
    })
</script>

</body>
</html>
```

3、员工信息添加页面employee_add.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>add employee</title>
    <link rel="stylesheet" th:href="@{/static/css/index_work.css}">
</head>
<body>
<form th:action="@{/employee}" method="post">
    <table>
        <tr>
            <th colspan="2">添加员工</th>
        </tr>
        <tr>
            <td>员工姓名</td>
            <td>
                <input type="text" name="empName">
            </td>
        </tr>
        <tr>
            <td>年龄</td>
            <td>
                <input type="text" name="age">
            </td>
        </tr>
        <tr>
            <td>性别</td>
```

```
            <td>
                <input type="radio" name="sex" value="男">男
                <input type="radio" name="sex" value="女">女
            </td>
        </tr>
        <tr>
            <td>邮箱</td>
            <td>
                <input type="text" name="email">
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" value="添加">
            </td>
        </tr>
    </table>

</form>
</body>
</html>
```

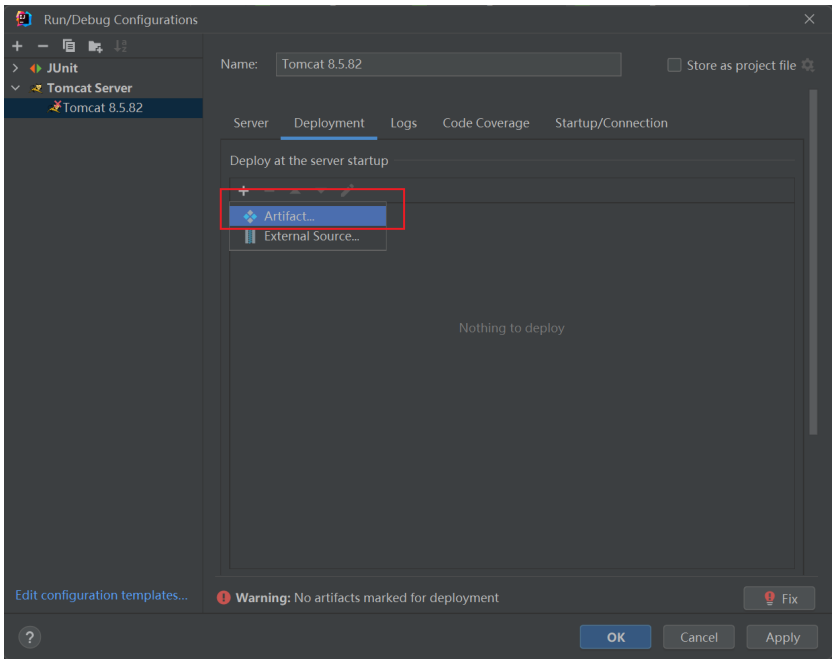
4、员工信息修改页面employee_update.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>add employee</title>
    <link rel="stylesheet" th:href="@{/static/css/index_work.css}">
</head>
<body>
<form th:action="@{/employee}" method="post">
    <input type="hidden" name="_method" value="put">
    <input type="hidden" name="empId" th:value="${employee.empId}">
    <table>
        <tr>
            <th colspan="2">修改员工信息</th>
        </tr>
        <tr>
            <td>员工姓名</td>
            <td>
                <input type="text" name="empName"
th:value="${employee.empName}">
            </td>
        </tr>
        <tr>
            <td>年龄</td>
            <td>
                <input type="text" name="age" th:value="${employee.age}">
            </td>
```

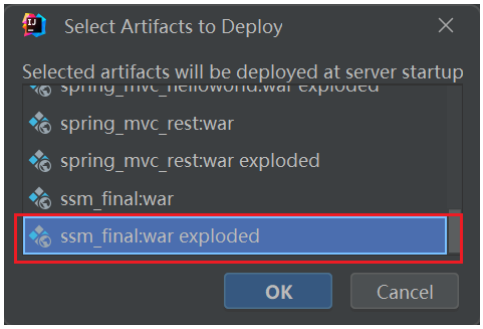
```
        </tr>

        <tr>
            <td>性别</td>
            <td>
                <input type="radio" name="sex" value="男"
th:field="${employee.sex}">男
                <input type="radio" name="sex" value="女"
th:field="${employee.sex}">女
            </td>
        </tr>
        <tr>
            <td>邮箱</td>
            <td>
                <input type="text" name="email"
th:value="${employee.email}">
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" value="修改">
            </td>
        </tr>
    </table>

</form>
</body>
</html>
```

7.5、部署到tomcat中

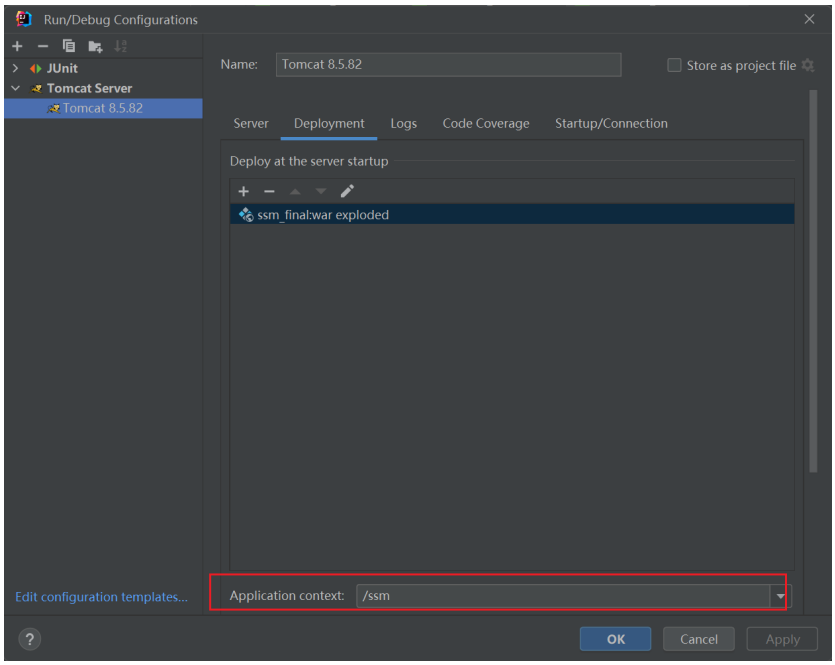
1、添加工程到tomcat中



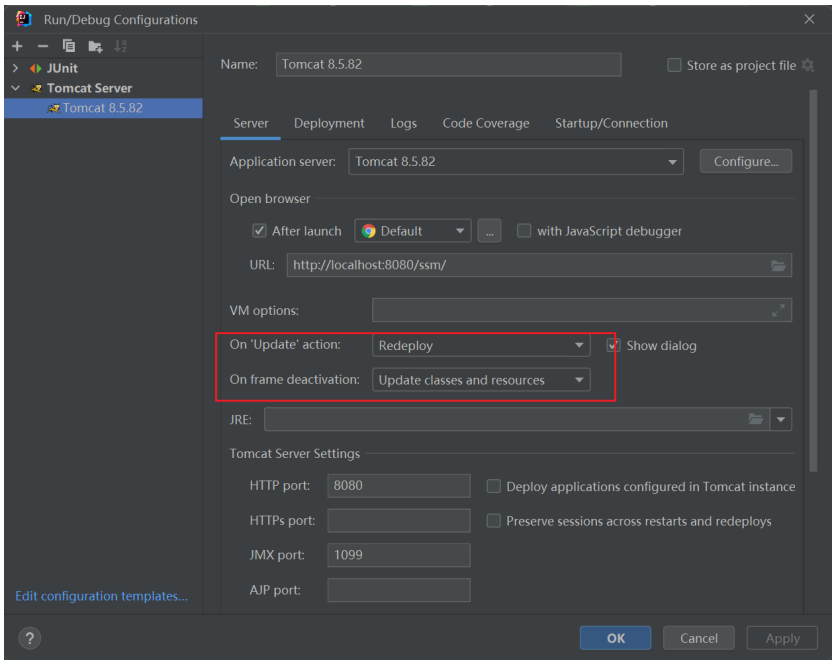
2、选择当前工程



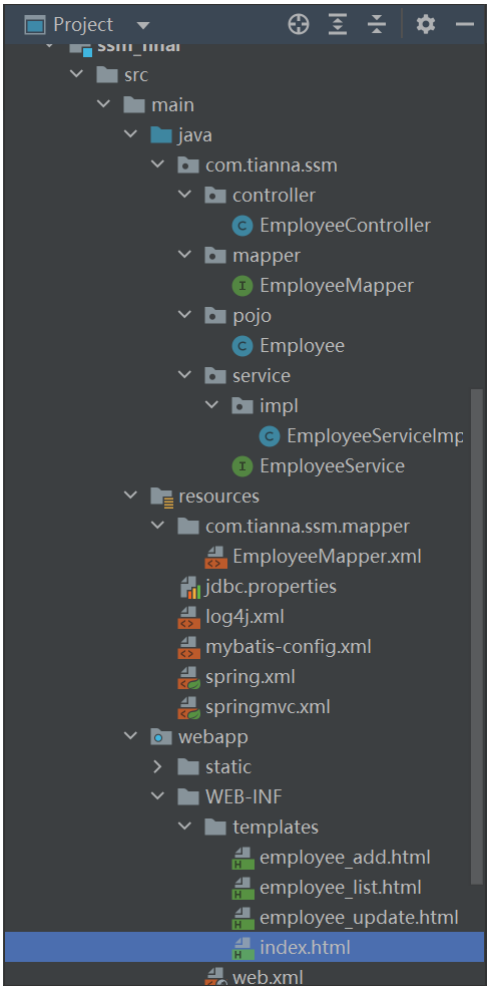
3、将上下问路径修改短一点(根据自己需要)



4、在Server一栏中，将On 'Update' action选择为Redeploy(重新部署)。将On frame deactivation选择为Update classes and resources（即当IDEA窗口失去焦点时，执行更新类和资源的操作）。



7.6、项目整体结构


Project
src
main
java
com.tianna.ssm
controller
EmployeeController
mapper
EmployeeMapper
pojo
Employee
service
impl
EmployeeServiceImp
EmployeeService
resources
com.tianna.ssm.mapper
EmployeeMapper.xml
jdbc.properties
log4j.xml
mybatis-config.xml
spring.xml
springmvc.xml
webapp
static
WEB-INF
templates
employee_add.html
employee_list.html
employee_update.html
index.html
web.xml